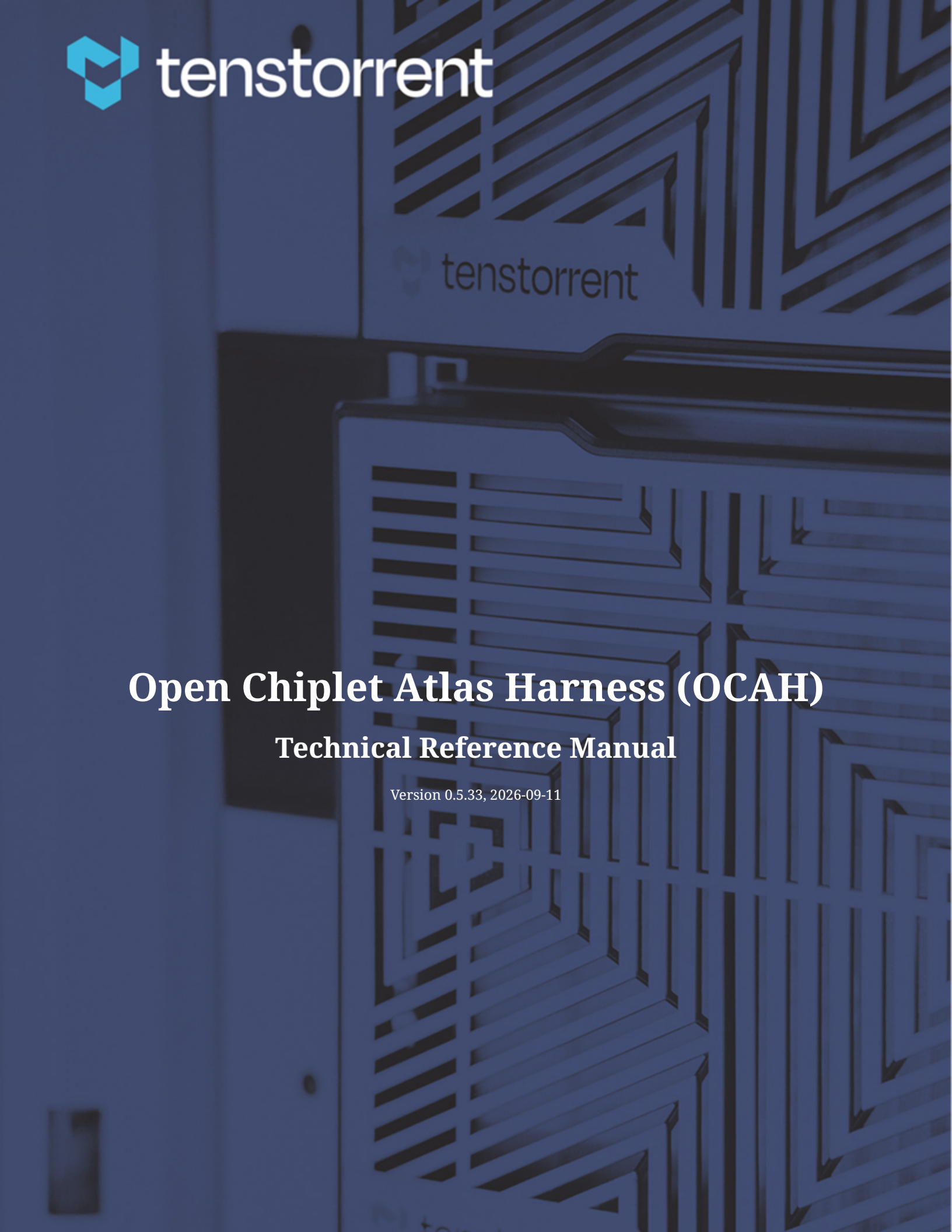


# Open Chiplet Atlas Harness (OCAH)

## Technical Reference Manual

Version 0.5.33, 2026-09-11

# Table of Contents


Documentation Overview . . . 1
List of Figures . . . 2
List of Tables . . . 6
1. OCAH Architecture & Design . . . 77
1.1. Platform Overview . . . 77
1.1.1. OCAH High-Level Features . . . 77
1.1.2. Top Block Diagram . . . 77
1.2. OCAH Memory Map . . . 78
1.2.1. Decode Response Codes . . . 79
1.3. Power Domains . . . 80
1.4. Clock Domains . . . 80
1.4.1. Reference Counter . . . 82
1.5. Reset . . . 82
1.5.1. IC_RESET TDR . . . 83
1.6. RTL Assertions . . . 83
2. OCAH Platform Components . . . 85
2.1. System Management Unit (SMU) . . . 85
2.1.1. Composition . . . 85
2.1.2. AXI Crossbar . . . 86
2.1.3. Clocks and Resets . . . 88
2.1.4. Lifecycle and Feature Control . . . 90
2.1.5. Security Processor (SEP) . . . 91
2.1.5.1. Overview . . . 91
Responsibilities . . . 93
Threat Model . . . 93
Attack Countermeasures . . . 93
Clocks and Resets . . . 93
CPU . . . 93
Boot ROM . . . 94
Interconnect . . . 94
Interrupts . . . 94
Peripherals . . . 94
Security and Lifecycle . . . 94
Cryptographic Subsystem . . . 94
Memory Map . . . 94
2.1.5.2. Threat Model . . . 94
Software Attacks . . . 94
Physical Attacks . . . 95
Supply-Chain Attacks . . . 95
Limits of Protection . . . 95
2.1.5.3. Attack Countermeasures . . . 95
Debug and DFT Security Considerations . . . 96

Debug and Scan Dump . . . 96
Class 2 Asset Manifest and Naming . . . 96
Secure Test Mode (SECURE_TM) . . . 97
2.1.5.4. Clocks and Resets . . . 98
Clock Distribution . . . 98
Reset Architecture . . . 99
SEP Reset Controller . . . 100
Software Reset Register . . . 102
Isolation and Reset Sequencing . . . 103
Resetting the Key Manager . . . 105
Resetting the TRNG . . . 105
Memory Map and Register Reference . . . 106
Memory Map . . . 106
Detailed Register Map . . . 106
SEP Reset Control Address Map . . . 106
SW_RESET_N . . . 106
2.1.5.5. CPU . . . 106
Processor Core . . . 108
Local Memory . . . 108
Dual Scratch Register Banks . . . 108
Scratch Register Reference . . . 109
SEP Scratch Address Map . . . 109
SCRATCH[8] . . . 109
Reset and Boot . . . 109
Configuration Straps . . . 109
Reset Vector . . . 109
System Interfaces . . . 110
DMA Interface . . . 110
Debug . . . 110
Alarm Aggregation Processor (AGP) . . . 110
Memory Map and Register Reference . . . 110
Memory Map . . . 110
Detailed Register Map . . . 110
SEP CPU Control Address Map . . . 110
CLOCK_GATE_CTRL . . . 112
REFERENCE_COUNTER . . . 112
TIMEOUT_INTERRUPT . . . 112
PKA_CTRL . . . 112
TIMEOUT_COUNT_DMA . . . 113
TIMEOUT_COUNT_SYS_IN . . . 113
TIMEOUT_COUNT_MAILBOX_INBOUND . . . 113
TIMEOUT_COUNT_MAILBOX_OUTBOUND . . . 113
TIMEOUT_COUNT_ENTROPY_WRITE . . . 113
TIMEOUT_COUNT_ENTROPY_READ . . . 113

TIMEOUT_COUNT_FILTER_OUT . . . 114
TIMEOUT_COUNT_ALIAS_REMAP . . . 114
TIMEOUT_ENABLE . . . 114
TIMEOUT_CLEAR . . . 114
TIMEOUT_MODE . . . 114
SEP_TEST_CTRL . . . 114
SEP_GLOBAL_BASE_ADDR . . . 115
SEP_LOCAL_BASE_ADDR . . . 115
SEP_REGION_SIZE . . . 115
SMU_GLOBAL_BASE_ADDR . . . 115
SMU_REGION_SIZE . . . 115
SMC_FUSE_SENSE_STATUS . . . 115
SEP_FUSE_SENSE_STATUS . . . 116
SEP_SW_DEBUG . . . 116
SEP_NMI_VEC . . . 116
SEP_NMI_VEC_LOCK . . . 116
EXT_TRNG_SRC_SEL . . . 116
EXT_TRNG_SRC_SEL_LOCK . . . 117
KM_WIPE_CTRL . . . 117
DMA_BUS_ERR_STATUS . . . 117
DMA_BUS_ERR_CLEAR . . . 117
PERIPH_BUS_ERR_STATUS . . . 117
PERIPH_BUS_ERR_CLEAR . . . 118
SEP_VERSION_ID . . . 118
2.1.5.6. Boot ROM . . . 118
ROM Architecture . . . 118
Hardware Configuration and Integrity . . . 118
Boot Flow . . . 119
BL0 Secure Boot and BL1 Execution Unlock . . . 119
Firmware Reference . . . 119
2.1.5.7. Interconnect . . . 119
Topology . . . 119
Top Level AXI4 . . . 119
Fabric Topology . . . 121
Interfaces . . . 124
Address Remapping . . . 125
Filtering and Protection . . . 125
Routing . . . 125
Transaction Routing . . . 126
Memory Map and Register Reference . . . 127
Memory Map . . . 127
Detailed Register Map . . . 127
2.1.5.8. Interrupts . . . 127
Controllers . . . 127

Sources and Routing . . . 127
SEP CPU Interrupt Vector Map . . . 127
Non-PIC Interrupt Paths . . . 129
NMI — Watchdog Timer Bark . . . 129
Timer and Software Interrupts . . . 129
Memory ECC Errors . . . 129
Correctable (single-bit) errors . . . 129
Uncorrectable (double-bit) errors . . . 130
Debug bus mirror . . . 130
Software Handling . . . 130
Memory Map and Register Reference . . . 131
Memory Map . . . 131
Detailed Register Map . . . 131
EL2_PIC . . . 131
meipl[256] . . . 132
meip[8] . . . 132
meie[256] . . . 132
mpiccfg . . . 132
meigwctrl[256] . . . 132
meigwclr[256] . . . 132
2.1.5.9. Peripherals . . . 133
Direct Memory Access (DMA) . . . 133
OCAH Modifications . . . 133
Configuration Taken at Each Start . . . 134
Zero Fill . . . 135
Completion Reporting . . . 135
Future Enhancements . . . 135
Memory Map and Register Reference . . . 135
Memory Map . . . 136
Detailed Register Map . . . 136
DMA Controller . . . 136
INTR_STATE . . . 137
INTR_ENABLE . . . 137
INTR_TEST . . . 137
ALERT_TEST . . . 137
SRC_ADDR_LO . . . 137
SRC_ADDR_HI . . . 138
DST_ADDR_LO . . . 138
DST_ADDR_HI . . . 138
ADDR_SPACE_ID . . . 138
ENABLED_MEMORY_RANGE_BASE . . . 138
ENABLED_MEMORY_RANGE_LIMIT . . . 138
RANGE_VALID . . . 139
RANGE_REGWEN . . . 139

CFG_REGWEN . . . 139
TOTAL_DATA_SIZE . . . 139
CHUNK_DATA_SIZE . . . 139
TRANSFER_WIDTH . . . 140
CONTROL . . . 140
SRC_CONFIG . . . 140
DST_CONFIG . . . 141
STATUS . . . 141
ERROR_CODE . . . 141
SHA2_DIGEST_0[16] . . . 142
HANDSHAKE_INTR_ENABLE . . . 142
CLEAR_INTR_SRC . . . 142
CLEAR_INTR_BUS . . . 142
INTR_SRC_ADDR_0[11] . . . 142
INTR_SRC_WR_VAL_0[11] . . . 142
Mailboxes . . . 143
Peripheral Subsystem . . . 144
Serial Peripheral Interface (SPI) . . . 145
Use Cases . . . 145
Features . . . 146
Integration and Register Access . . . 146
Interrupts . . . 146
DMA Trigger . . . 147
Core Reference . . . 147
Memory Map and Register Reference . . . 147
Memory Map . . . 147
Detailed Register Map . . . 147
SPI Host . . . 147
INTR_STATE . . . 147
INTR_ENABLE . . . 148
INTR_TEST . . . 148
ALERT_TEST . . . 148
CONTROL . . . 148
STATUS . . . 149
CONFIGOPTS . . . 149
CSID . . . 150
COMMAND . . . 150
ERROR_ENABLE . . . 151
ERROR_STATUS . . . 151
EVENT_ENABLE . . . 151
Watchdog Timer (WDT) . . . 152
Bus Interface . . . 152
Clock, Reset, and Debug Behavior . . . 152
Bark Interrupt and Bite Reset . . . 152

Programming Reference . . . 152
Memory Map and Register Reference . . . 152
Memory Map . . . 152
Detailed Register Map . . . 152
Always-On Timer . . . 152
ALERT_TEST . . . 153
WKUP_CTRL . . . 153
WKUP_THOLD_HI . . . 153
WKUP_THOLD_LO . . . 153
WKUP_COUNT_HI . . . 154
WKUP_COUNT_LO . . . 154
WDOG_REGWEN . . . 154
WDOG_CTRL . . . 154
WDOG_BARK_THOLD . . . 154
WDOG_BITE_THOLD . . . 154
WDOG_COUNT . . . 154
INTR_STATE . . . 155
INTR_TEST . . . 155
WKUP_CAUSE . . . 155
2.1.5.10. Security and Lifecycle . . . 155
OTP Fuse Controller . . . 155
Fuse Sensing and Boot Sequencing . . . 158
Shadow Register and JTAG Access Control . . . 159
Fuse Programming and RMA Authorization . . . 159
Programming Voltage . . . 159
Shadow Register Updates and Test-Mode Protection . . . 159
Fuse Fields . . . 160
Anti-Rollback Security-Version Flags . . . 163
RTL Integration . . . 163
Memory Map and Register Reference . . . 163
Memory Map . . . 163
Detailed Register Map . . . 164
SEP eFuse Map Address Map . . . 164
LOCKS . . . 165
LOCKS_SPARE . . . 167
LC_STATE . . . 168
SBOOT_DIS . . . 168
TRANSIENT_RMA_EN . . . 168
SIP_DIS . . . 168
SYS_DIS . . . 169
RMA_SIP_TOKEN_DIGEST . . . 170
RMA_CHIPLET_TOKEN_DIGEST . . . 170
CLASS_KEY . . . 170
CHIPLET_PUBK_REVOKE . . . 170

BL1_VERSION . . . 170
BL2_VERSION . . . 171
CHIPLET_UID . . . 171
SIP_PUBK_HASH0 . . . 171
SIP_UID . . . 171
SYS_PUBK_HASH . . . 171
SYS_UID . . . 171
STATUS_RPT . . . 171
ROM_CTL . . . 172
SYSCLK_FREQ_MHZ . . . 172
CHIPLET_PUBK_HASH0 . . . 172
CHIPLET_PUBK_HASH1 . . . 172
REQUIRED_SIGNERS . . . 172
REQUIRED_ALGS . . . 173
CHIPLET_PUBK_PQC_HASH0 . . . 173
CHIPLET_PUBK_PQC_HASH1 . . . 173
SIP_PUBK_PQC_HASH0 . . . 173
SYS_PUBK_PQC_HASH . . . 173
SIP_PUBK_HASH1 . . . 173
SIP_PUBK_PQC_HASH1 . . . 174
SEP_CHIPLET_ID . . . 174
SEP_SIP_ID . . . 174
SEP_SYS_ID . . . 174
spare0 . . . 174
spare1 . . . 174
spare2 . . . 174
spare3 . . . 175
spare4 . . . 175
spare5 . . . 175
spare6 . . . 175
spare7 . . . 175
spare8 . . . 175
Token Processing . . . 176
Token Generation, Storage and Provisioning . . . 176
Token Matching Mechanism and Attack Countermeasures . . . 176
Token Submission . . . 176
Matching Logic Countermeasures . . . 177
Post Token Matching Handling . . . 178
Security Disable (SEC_DIS) . . . 178
Token Activation . . . 179
Fuse Sensing and the Shadow Map . . . 179
Reset Persistence . . . 179
Silicon Revision Scope . . . 179
Implementation Requirements . . . 179

Life Cycle Controller (LCC) . . . 180
Lifecycle States . . . 181
Multiple Security Domains . . . 183
Debug and Functional Feature Control . . . 184
Disable Vector Format (SIP_DIS / SYS_DIS) . . . 184
DIS Vector Ownership: Predecessor-Programs-Disables . . . 185
Combining Rule: OR then Invert . . . 186
When Each DIS Vector Applies . . . 186
Overrides and Final Gates (SEC_DIS) . . . 186
Per-LC-state Feature Control Profile . . . 187
In-Field Debug Demotion . . . 187
Programming and Locking . . . 188
Reset Behavior . . . 188
Asset and Scan-Chain Protection . . . 188
Asset Classes . . . 188
`secure_tm` and the Secure Scan Chain . . . 189
Feature-Bit to Scan-Chain Mapping . . . 190
Debug and Test Port Path Gating . . . 190
Non-Primary Chiplets . . . 191
Adopter Extension . . . 192
Firmware Authorization . . . 192
Memory Map and Register Reference . . . 193
Memory Map . . . 193
Detailed Register Map . . . 193
SEP Lifecycle Control Address Map . . . 193
FEAT_CTRL . . . 193
DEMOTE_1 . . . 193
DEMOTE_2 . . . 193
Test Mode Entry . . . 194
Entry and Exit . . . 194
Protection While Active . . . 194
2.1.5.11. Cryptographic Subsystem . . . 194
OpenTitan Alert Termination . . . 197
Single-Beat Access Only . . . 198
Key Manager Key Delivery . . . 198
TRNG and DRBG . . . 199
Entropy Generation . . . 200
Distribution and Source Selection . . . 200
FIPS Provenance . . . 201
Entropy Access Paths . . . 201
Reset and Recovery Boundaries . . . 201
Scan Protection Boundary . . . 202
DRBG Deterministic Random Bit Generator . . . 202
Parameters . . . 202

External Interfaces . . . 202
Seed Assembly . . . 203
OpenTitan CSRNG and EDN . . . 203
Bus Protocol Adaptation . . . 203
Memory Map and Register Reference . . . 204
Memory Map . . . 204
Detailed Register Map . . . 204
CSRNG Registers . . . 204
Cryptographically Secure Random Number Generator . . . 204
INTR_STATE . . . 205
INTR_ENABLE . . . 205
INTR_TEST . . . 205
ALERT_TEST . . . 206
REGWEN . . . 206
CTRL . . . 206
CMD_REQ . . . 206
RESEED_INTERVAL . . . 206
RESEED_COUNTER_0 . . . 207
RESEED_COUNTER_1 . . . 207
RESEED_COUNTER_2 . . . 207
SW_CMD_STS . . . 207
GENBITS_VLD . . . 208
GENBITS . . . 208
INT_STATE_READ_ENABLE . . . 208
INT_STATE_READ_ENABLE_REGWEN . . . 208
INT_STATE_NUM . . . 208
INT_STATE_VAL . . . 209
FIPS_FORCE . . . 209
HW_EXC_STS . . . 209
RECOV_ALERT_STS . . . 210
ERR_CODE . . . 210
ERR_CODE_TEST . . . 211
MAIN_SM_STATE . . . 211
EDN Registers . . . 211
Entropy Distribution Network . . . 211
INTR_STATE . . . 212
INTR_ENABLE . . . 212
INTR_TEST . . . 212
ALERT_TEST . . . 213
REGWEN . . . 213
CTRL . . . 213
BOOT_INS_CMD . . . 214
BOOT_GEN_CMD . . . 214
SW_CMD_REQ . . . 214

SW_CMD_STS . . . 214
HW_CMD_STS . . . 215
RESEED_CMD . . . 215
GENERATE_CMD . . . 216
MAX_NUM_REQS_BETWEEN_RESEEDS . . . 216
RECOV_ALERT_STS . . . 216
ERR_CODE . . . 217
ERR_CODE_TEST . . . 218
MAIN_SM_STATE . . . 218
Entropy Source . . . 218
Noise Generation . . . 219
Decorrelation and BIW Mixing . . . 221
Health Testing . . . 221
Health Test Watermark . . . 222
Persistent Failure and AlertHang . . . 222
SHA-256 Whitening . . . 222
Output FIFO . . . 222
FIFO Access Gate . . . 223
Reset . . . 223
Debug Observation Pin . . . 224
Memory Map and Register Reference . . . 224
Memory Map . . . 224
Detailed Register Map . . . 224
Entropy Source Address Map . . . 224
COMPONENT_ID . . . 226
CTRL . . . 226
DEBUG_CTRL . . . 227
INTR_STATUS . . . 227
INTR_ENABLE . . . 228
INTR_TEST . . . 228
SHA256_STATUS . . . 229
FIFO_CTRL . . . 229
FIFO_STATUS . . . 229
FIFO_RDATA . . . 229
HEALTH_TEST_CTRL . . . 230
HEALTH_TEST_WINDOW_SIZE . . . 230
MARKOV_TEST_PROB_THRESHOLDS . . . 230
HEALTH_TEST_STATUS . . . 230
REPETITION_TEST_COUNT . . . 230
APT_PATTERN_COUNT_1BIT . . . 230
APT_PATTERN_COUNT_2BIT . . . 231
APT_PROPORTION_1BIT . . . 231
APT_PROPORTION_LO . . . 231
MARKOV_TEST_COUNTS_0 . . . 231

RING_OSC_ENABLE . . . 231
RING_OSC_TUNE . . . 232
RING_OSC_CTRL . . . 232
DECORRELATOR_CTRL . . . 232
DECORRELATOR_MASK . . . 232
MAIN_SM_STATUS . . . 232
GENERATOR_0_HEALTH_STATUS . . . 233
GENERATOR_1_HEALTH_STATUS . . . 233
GENERATOR_2_HEALTH_STATUS . . . 233
GENERATOR_3_HEALTH_STATUS . . . 233
GENERATOR_4_HEALTH_STATUS . . . 233
GENERATOR_5_HEALTH_STATUS . . . 233
GENERATOR_6_HEALTH_STATUS . . . 234
GENERATOR_7_HEALTH_STATUS . . . 234
GENERATOR_8_HEALTH_STATUS . . . 234
GENERATOR_9_HEALTH_STATUS . . . 234
GENERATOR_10_HEALTH_STATUS . . . 234
GENERATOR_11_HEALTH_STATUS . . . 234
GENERATOR_0_SAMPLE_CLK_CONFIG . . . 234
GENERATOR_1_SAMPLE_CLK_CONFIG . . . 235
GENERATOR_2_SAMPLE_CLK_CONFIG . . . 235
GENERATOR_3_SAMPLE_CLK_CONFIG . . . 235
GENERATOR_4_SAMPLE_CLK_CONFIG . . . 235
GENERATOR_5_SAMPLE_CLK_CONFIG . . . 235
GENERATOR_6_SAMPLE_CLK_CONFIG . . . 236
GENERATOR_7_SAMPLE_CLK_CONFIG . . . 236
GENERATOR_8_SAMPLE_CLK_CONFIG . . . 236
GENERATOR_9_SAMPLE_CLK_CONFIG . . . 236
GENERATOR_10_SAMPLE_CLK_CONFIG . . . 236
GENERATOR_11_SAMPLE_CLK_CONFIG . . . 236
HT_WATERMARK_NUM . . . 237
HT_WATERMARK . . . 237
REPCNT_TOTAL_FAILS . . . 237
APT_HI_TOTAL_FAILS . . . 237
APT_LO_TOTAL_FAILS . . . 237
MARKOV_HI_TOTAL_FAILS . . . 237
MARKOV_LO_TOTAL_FAILS . . . 237
ALERT_SUMMARY_FAIL_COUNTS . . . 238
ALERT_FAIL_COUNTS . . . 238
FIPS_LOCK . . . 238
ALERT_THRESHOLD . . . 238
MIN_ENTROPY_H . . . 238
RECOMMENDED_THRESHOLDS . . . 238
BIW_OBS_CTRL . . . 239

BIW_OBS_STATUS . . . 239
BIW_OBS_RDATA . . . 239
NOISE_OBS_CTRL . . . 239
NOISE_OBS_STATUS . . . 239
NOISE_OBS_RDATA . . . 240
Entropy Pool . . . 240
Access Policy . . . 240
Memory Map and Register Reference . . . 240
Memory Map . . . 240
Detailed Register Map . . . 240
SEP Entropy Pool . . . 240
STATUS . . . 241
IRQ_CAUSE . . . 241
DATA . . . 241
AES Cipher . . . 241
OCAH Modifications . . . 242
Memory Map and Register Reference . . . 242
Memory Map . . . 242
Detailed Register Map . . . 243
AES Accelerator . . . 243
ALERT_TEST . . . 243
KEY_SHARE0[8] . . . 243
KEY_SHARE1[8] . . . 244
IV[4] . . . 244
DATA_IN[4] . . . 244
DATA_OUT[4] . . . 244
CTRL_SHADOWED . . . 244
CTRL_AUX_SHADOWED . . . 245
CTRL_AUX_REGWEN . . . 245
TRIGGER . . . 245
STATUS . . . 246
CTRL_GCM_SHADOWED . . . 246
HMAC-SHA2 . . . 246
OCAH Modifications . . . 247
Memory Map and Register Reference . . . 247
Memory Map . . . 247
Detailed Register Map . . . 248
HMAC Accelerator . . . 248
INTR_STATE . . . 248
INTR_ENABLE . . . 249
INTR_TEST . . . 249
ALERT_TEST . . . 249
CFG . . . 249
CMD . . . 250

STATUS . . . 251
ERR_CODE . . . 251
WIPE_SECRET . . . 251
KEY[32] . . . 251
DIGEST[16] . . . 251
MSG_LENGTH_LOWER . . . 251
MSG_LENGTH_UPPER . . . 252
KMAC-SHA3 . . . 252
OCAH Modifications . . . 252
Memory Map and Register Reference . . . 253
Memory Map . . . 253
Detailed Register Map . . . 253
KMAC Accelerator . . . 253
INTR_STATE . . . 254
INTR_ENABLE . . . 254
INTR_TEST . . . 254
ALERT_TEST . . . 254
CFG_REGWEN . . . 254
CFG_SHADOWED . . . 255
CMD . . . 256
STATUS . . . 256
ENTROPY_PERIOD . . . 257
ENTROPY_REFRESH_HASH_CNT . . . 258
ENTROPY_REFRESH_THRESHOLD_SHADOWED . . . 258
ENTROPY_SEED . . . 258
KEY_SHARE0[16] . . . 258
KEY_SHARE1[16] . . . 258
KEY_LEN . . . 258
PREFIX[11] . . . 258
ERR_CODE . . . 259
Adams Bridge (ABR) . . . 259
OCAH Modifications . . . 260
OTBN PKA Public Key Accelerator . . . 262
OCAH Modifications . . . 264
Memory Map and Register Reference . . . 264
Memory Map . . . 264
Detailed Register Map . . . 264
OpenTitan Big Number Accelerator . . . 264
INTR_STATE . . . 265
INTR_ENABLE . . . 265
INTR_TEST . . . 265
ALERT_TEST . . . 265
CMD . . . 265
CTRL . . . 266

STATUS . . . 266
ERR_BITS . . . 266
FATAL_ALERT_CAUSE . . . 267
INSN_CNT . . . 267
LOAD_CHECKSUM . . . 267
Key Manager (KM) . . . 267
Architecture . . . 268
Parameters . . . 270
Reset Conditioning . . . 270
Mailbox Transport . . . 271
Key Delivery . . . 273
Adams Bridge Key Sideload Interfaces . . . 274
Key Manager CPU Interrupt Processing . . . 275
Host Coordination . . . 275
Key and Policy Vault (KPV) . . . 277
KPV Scrambling . . . 278
Wipe State . . . 278
Scan and Scan-Dump Protection . . . 278
Key Manager Control and Status Registers (KMCSR) . . . 279
DRBG Sampler . . . 282
Reads, prefetch and register access . . . 282
Errors and reset . . . 283
OTP Inputs and Readout . . . 284
CRC Accelerator . . . 285
ROM and SRAM Error Detection . . . 286
Parity Error Protection . . . 286
SRAM Write Locking . . . 286
ROM Warm-Persistent SRAM Region . . . 287
Mutable Firmware Model . . . 287
SRAM Scrambling . . . 288
Parameters . . . 288
Port Interface . . . 288
Data Scrambling Algorithm . . . 288
Address Tweaking . . . 290
Address Scrambling . . . 290
Byte-Wise Scrambling Mode . . . 291
Instantiation Variants . . . 291
Integration in the Key Manager . . . 292
Firmware . . . 293
Memory Map and Register Reference . . . 293
Memory Map . . . 293
Detailed Register Map . . . 295
Address Summary Registers . . . 295
Key Manager Top-Level Address Map . . . 295

KM_WRITE_DATA . . . 307
KM_WRITE_SEPARATOR . . . 307
KM_READ_DATA . . . 308
KM_STATUS . . . 308
KM_IRQ_STATUS . . . 308
KM_IRQ_ENABLE . . . 309
KM_CTRL . . . 309
LOCKS . . . 309
LOCKS_SPARE . . . 312
LC_STATE . . . 312
SBOOT_DIS . . . 312
TRANSIENT_RMA_EN . . . 313
SIP_DIS . . . 313
SYS_DIS . . . 313
RMA_SIP_TOKEN_DIGEST . . . 314
RMA_CHIPLET_TOKEN_DIGEST . . . 314
CLASS_KEY . . . 314
CHIPLET_PUBK_REVOKE . . . 314
BL1_VERSION . . . 315
BL2_VERSION . . . 315
CHIPLET_UID . . . 315
SIP_PUBK_HASH0 . . . 315
SIP_UID . . . 315
SYS_PUBK_HASH . . . 316
SYS_UID . . . 316
STATUS_RPT . . . 316
ROM_CTL . . . 316
SYSCLK_FREQ_MHZ . . . 316
CHIPLET_PUBK_HASH0 . . . 316
CHIPLET_PUBK_HASH1 . . . 317
REQUIRED_SIGNERS . . . 317
REQUIRED_ALGS . . . 317
CHIPLET_PUBK_PQC_HASH0 . . . 317
CHIPLET_PUBK_PQC_HASH1 . . . 317
SIP_PUBK_PQC_HASH0 . . . 317
SYS_PUBK_PQC_HASH . . . 318
SIP_PUBK_HASH1 . . . 318
SIP_PUBK_PQC_HASH1 . . . 318
SEP_CHIPLET_ID . . . 318
SEP_SIP_ID . . . 318
SEP_SYS_ID . . . 318
spare0 . . . 319
spare1 . . . 319
spare2 . . . 319

spare3 . . . 319
spare4 . . . 319
spare5 . . . 319
spare6 . . . 319
spare7 . . . 320
spare8 . . . 320
EFUSE_INTERFACE_CTRL_STATUS . . . 320
EFUSE_PROGRAM_CTRL . . . 320
EFUSE_READ_CTRL . . . 321
EFUSE_PROGRAM_INTERFACE_READ_DATA . . . 321
EFUSE_READ_INTERFACE_READ_DATA . . . 321
EFUSE_READ_REQ_TIMEOUT . . . 321
EFUSE_PROGRAM_REQ_TIMEOUT . . . 322
RMA_SIP_TOKEN_I[8] . . . 322
RMA_CHIPLET_TOKEN_I[8] . . . 322
SEC_DISABLE_TOKEN_I[8] . . . 322
TOKEN_EOP . . . 322
RMA_SIP_TOKEN_MATCH . . . 322
RMA_CHIPLET_TOKEN_MATCH . . . 323
SEC_DISABLE_TOKEN_MATCH . . . 323
TOKEN_MATCH_FAULT . . . 323
KEY_ENTRY[64].WORD[16] . . . 323
CTRL[64] . . . 323
KPV_SCRAMBLER_KEY . . . 324
KPV_SCRAMBLER_CTRL . . . 324
VERSION . . . 324
CTRL . . . 324
SOFT_RST_CODE . . . 325
kmcsr.IRQ_STATUS . . . 325
kmcsr.IRQ_ENABLE . . . 325
SCRAMBLER_KEY . . . 326
SCRAMBLER_CTRL . . . 326
SRAM_LOCK . . . 326
IRQ_SET . . . 326
SRAM_WRITE_LOCK_VIOLATION . . . 327
RECOVERABLE_ERR . . . 327
BOOT_STATUS . . . 327
OTP_LIFE_CYCLE . . . 327
OTP_DEMOTION_STATE . . . 328
SRAM_EXEC_MODE . . . 328
IRQ_ENTRY_ADDR . . . 328
IRQ_ENTRY_LOCK . . . 328
VUART_TX . . . 328
VUART_RX . . . 328

VUART_STATUS . . . 329
TB_RESULT . . . 329
TB_SIGNATURE . . . 329
TB_ERRCODE . . . 329
TB_SUBTEST . . . 329
TB_CMD . . . 329
TB_CMD_ARG . . . 330
TB_CMD_STATUS . . . 330
TB_CMD_RESULT . . . 330
DEBUG . . . 330
OTP_CHIPLET_UID_VAL_0 . . . 330
OTP_CHIPLET_UID_VAL_1 . . . 330
OTP_CHIPLET_UID_VAL_2 . . . 330
OTP_CHIPLET_UID_VAL_3 . . . 330
OTP_CHIPLET_UID_VAL_4 . . . 331
OTP_CHIPLET_UID_VAL_5 . . . 331
OTP_CHIPLET_UID_VAL_6 . . . 331
OTP_CHIPLET_UID_VAL_7 . . . 331
OTP_CHIPLET_UID_CPL_0 . . . 331
OTP_CHIPLET_UID_CPL_1 . . . 331
OTP_CHIPLET_UID_CPL_2 . . . 331
OTP_CHIPLET_UID_CPL_3 . . . 332
OTP_CHIPLET_UID_CPL_4 . . . 332
OTP_CHIPLET_UID_CPL_5 . . . 332
OTP_CHIPLET_UID_CPL_6 . . . 332
OTP_CHIPLET_UID_CPL_7 . . . 332
OTP_SIP_UID_VAL_0 . . . 332
OTP_SIP_UID_VAL_1 . . . 332
OTP_SIP_UID_VAL_2 . . . 332
OTP_SIP_UID_VAL_3 . . . 333
OTP_SIP_UID_VAL_4 . . . 333
OTP_SIP_UID_VAL_5 . . . 333
OTP_SIP_UID_VAL_6 . . . 333
OTP_SIP_UID_VAL_7 . . . 333
OTP_SIP_UID_CPL_0 . . . 333
OTP_SIP_UID_CPL_1 . . . 333
OTP_SIP_UID_CPL_2 . . . 334
OTP_SIP_UID_CPL_3 . . . 334
OTP_SIP_UID_CPL_4 . . . 334
OTP_SIP_UID_CPL_5 . . . 334
OTP_SIP_UID_CPL_6 . . . 334
OTP_SIP_UID_CPL_7 . . . 334
OTP_SYS_UID_VAL_0 . . . 334
OTP_SYS_UID_VAL_1 . . . 334

OTP_SYS_UID_VAL_2 . . . 335
OTP_SYS_UID_VAL_3 . . . 335
OTP_SYS_UID_VAL_4 . . . 335
OTP_SYS_UID_VAL_5 . . . 335
OTP_SYS_UID_VAL_6 . . . 335
OTP_SYS_UID_VAL_7 . . . 335
OTP_SYS_UID_CPL_0 . . . 335
OTP_SYS_UID_CPL_1 . . . 336
OTP_SYS_UID_CPL_2 . . . 336
OTP_SYS_UID_CPL_3 . . . 336
OTP_SYS_UID_CPL_4 . . . 336
OTP_SYS_UID_CPL_5 . . . 336
OTP_SYS_UID_CPL_6 . . . 336
OTP_SYS_UID_CPL_7 . . . 336
OTP_CLASS_KEY_VAL_0 . . . 336
OTP_CLASS_KEY_VAL_1 . . . 337
OTP_CLASS_KEY_VAL_2 . . . 337
OTP_CLASS_KEY_VAL_3 . . . 337
OTP_CLASS_KEY_VAL_4 . . . 337
OTP_CLASS_KEY_VAL_5 . . . 337
OTP_CLASS_KEY_VAL_6 . . . 337
OTP_CLASS_KEY_VAL_7 . . . 337
OTP_CLASS_KEY_CPL_0 . . . 338
OTP_CLASS_KEY_CPL_1 . . . 338
OTP_CLASS_KEY_CPL_2 . . . 338
OTP_CLASS_KEY_CPL_3 . . . 338
OTP_CLASS_KEY_CPL_4 . . . 338
OTP_CLASS_KEY_CPL_5 . . . 338
OTP_CLASS_KEY_CPL_6 . . . 338
OTP_CLASS_KEY_CPL_7 . . . 338
OTP_READ_LOCK . . . 339
OTP_CHANGE_STATUS . . . 339
OTP_READ_LOCK_COLD . . . 339
OTP_SEP_CHIPLET_ID_VAL_0 . . . 340
OTP_SEP_CHIPLET_ID_VAL_1 . . . 340
OTP_SEP_CHIPLET_ID_VAL_2 . . . 340
OTP_SEP_CHIPLET_ID_VAL_3 . . . 340
OTP_SEP_CHIPLET_ID_VAL_4 . . . 340
OTP_SEP_CHIPLET_ID_VAL_5 . . . 341
OTP_SEP_CHIPLET_ID_VAL_6 . . . 341
OTP_SEP_CHIPLET_ID_VAL_7 . . . 341
OTP_SEP_CHIPLET_ID_CPL_0 . . . 341
OTP_SEP_CHIPLET_ID_CPL_1 . . . 341
OTP_SEP_CHIPLET_ID_CPL_2 . . . 341

OTP_SEP_CHIPLET_ID_CPL_3 . . . 341
OTP_SEP_CHIPLET_ID_CPL_4 . . . 341
OTP_SEP_CHIPLET_ID_CPL_5 . . . 342
OTP_SEP_CHIPLET_ID_CPL_6 . . . 342
OTP_SEP_CHIPLET_ID_CPL_7 . . . 342
OTP_SEP_SIP_ID_VAL_0 . . . 342
OTP_SEP_SIP_ID_VAL_1 . . . 342
OTP_SEP_SIP_ID_VAL_2 . . . 342
OTP_SEP_SIP_ID_VAL_3 . . . 342
OTP_SEP_SIP_ID_VAL_4 . . . 343
OTP_SEP_SIP_ID_VAL_5 . . . 343
OTP_SEP_SIP_ID_VAL_6 . . . 343
OTP_SEP_SIP_ID_VAL_7 . . . 343
OTP_SEP_SIP_ID_CPL_0 . . . 343
OTP_SEP_SIP_ID_CPL_1 . . . 343
OTP_SEP_SIP_ID_CPL_2 . . . 343
OTP_SEP_SIP_ID_CPL_3 . . . 343
OTP_SEP_SIP_ID_CPL_4 . . . 344
OTP_SEP_SIP_ID_CPL_5 . . . 344
OTP_SEP_SIP_ID_CPL_6 . . . 344
OTP_SEP_SIP_ID_CPL_7 . . . 344
OTP_SEP_SYS_ID_VAL_0 . . . 344
OTP_SEP_SYS_ID_VAL_1 . . . 344
OTP_SEP_SYS_ID_VAL_2 . . . 344
OTP_SEP_SYS_ID_VAL_3 . . . 345
OTP_SEP_SYS_ID_VAL_4 . . . 345
OTP_SEP_SYS_ID_VAL_5 . . . 345
OTP_SEP_SYS_ID_VAL_6 . . . 345
OTP_SEP_SYS_ID_VAL_7 . . . 345
OTP_SEP_SYS_ID_CPL_0 . . . 345
OTP_SEP_SYS_ID_CPL_1 . . . 345
OTP_SEP_SYS_ID_CPL_2 . . . 345
OTP_SEP_SYS_ID_CPL_3 . . . 346
OTP_SEP_SYS_ID_CPL_4 . . . 346
OTP_SEP_SYS_ID_CPL_5 . . . 346
OTP_SEP_SYS_ID_CPL_6 . . . 346
OTP_SEP_SYS_ID_CPL_7 . . . 346
DATA . . . 346
CFG . . . 346
STATUS . . . 347
PREFETCH_DATA . . . 347
KEY_SHARE0[12] . . . 347
KEY_SHARE1[12] . . . 347
otbn_wrapper_key.KEY_CTRL . . . 347

aes_wrapper_key.KEY_SHARE0[8] . . . 348
aes_wrapper_key.KEY_SHARE1[8] . . . 348
aes_wrapper_key.KEY_CTRL . . . 348
kmac_wrapper_key.KEY_SHARE0[8] . . . 348
kmac_wrapper_key.KEY_SHARE1[8] . . . 348
kmac_wrapper_key.KEY_CTRL . . . 348
hmac_wrapper_key.KEY_SHARE0[8] . . . 348
hmac_wrapper_key.KEY_SHARE1[8] . . . 349
hmac_wrapper_key.KEY_CTRL . . . 349
abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] . . . 349
abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] . . . 349
abr_wrapper_key.MLDSA_SEED.KEY_CTRL . . . 349
abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] . . . 349
abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] . . . 349
abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL . . . 350
abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] . . . 350
abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] . . . 350
abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL . . . 350
abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] . . . 350
abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] . . . 350
abr_wrapper_key.MLKEM_MSG.KEY_CTRL . . . 350
KEY[8] . . . 351
abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL . . . 351
abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_STATUS . . . 351
abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_ENABLE . . . 351
Control and Status Registers . . . 351
Key Manager Control and Status Registers . . . 351
VERSION . . . 359
CTRL . . . 359
SOFT_RST_CODE . . . 359
IRQ_STATUS . . . 359
IRQ_ENABLE . . . 360
SCRAMBLER_KEY . . . 360
SCRAMBLER_CTRL . . . 361
SRAM_LOCK . . . 361
IRQ_SET . . . 361
SRAM_WRITE_LOCK_VIOLATION . . . 362
RECOVERABLE_ERR . . . 362
BOOT_STATUS . . . 362
OTP_LIFE_CYCLE . . . 362
OTP_DEMOTION_STATE . . . 362
SRAM_EXEC_MODE . . . 362
IRQ_ENTRY_ADDR . . . 363
IRQ_ENTRY_LOCK . . . 363

VUART_TX . . . 363
VUART_RX . . . 363
VUART_STATUS . . . 363
TB_RESULT . . . 363
TB_SIGNATURE . . . 364
TB_ERRCODE . . . 364
TB_SUBTEST . . . 364
TB_CMD . . . 364
TB_CMD_ARG . . . 364
TB_CMD_STATUS . . . 364
TB_CMD_RESULT . . . 364
DEBUG . . . 364
OTP_CHIPLET_UID_VAL_0 . . . 365
OTP_CHIPLET_UID_VAL_1 . . . 365
OTP_CHIPLET_UID_VAL_2 . . . 365
OTP_CHIPLET_UID_VAL_3 . . . 365
OTP_CHIPLET_UID_VAL_4 . . . 365
OTP_CHIPLET_UID_VAL_5 . . . 365
OTP_CHIPLET_UID_VAL_6 . . . 365
OTP_CHIPLET_UID_VAL_7 . . . 366
OTP_CHIPLET_UID_CPL_0 . . . 366
OTP_CHIPLET_UID_CPL_1 . . . 366
OTP_CHIPLET_UID_CPL_2 . . . 366
OTP_CHIPLET_UID_CPL_3 . . . 366
OTP_CHIPLET_UID_CPL_4 . . . 366
OTP_CHIPLET_UID_CPL_5 . . . 366
OTP_CHIPLET_UID_CPL_6 . . . 366
OTP_CHIPLET_UID_CPL_7 . . . 367
OTP_SIP_UID_VAL_0 . . . 367
OTP_SIP_UID_VAL_1 . . . 367
OTP_SIP_UID_VAL_2 . . . 367
OTP_SIP_UID_VAL_3 . . . 367
OTP_SIP_UID_VAL_4 . . . 367
OTP_SIP_UID_VAL_5 . . . 367
OTP_SIP_UID_VAL_6 . . . 368
OTP_SIP_UID_VAL_7 . . . 368
OTP_SIP_UID_CPL_0 . . . 368
OTP_SIP_UID_CPL_1 . . . 368
OTP_SIP_UID_CPL_2 . . . 368
OTP_SIP_UID_CPL_3 . . . 368
OTP_SIP_UID_CPL_4 . . . 368
OTP_SIP_UID_CPL_5 . . . 368
OTP_SIP_UID_CPL_6 . . . 369
OTP_SIP_UID_CPL_7 . . . 369

OTP_SYS_UID_VAL_0 . . . 369
OTP_SYS_UID_VAL_1 . . . 369
OTP_SYS_UID_VAL_2 . . . 369
OTP_SYS_UID_VAL_3 . . . 369
OTP_SYS_UID_VAL_4 . . . 369
OTP_SYS_UID_VAL_5 . . . 370
OTP_SYS_UID_VAL_6 . . . 370
OTP_SYS_UID_VAL_7 . . . 370
OTP_SYS_UID_CPL_0 . . . 370
OTP_SYS_UID_CPL_1 . . . 370
OTP_SYS_UID_CPL_2 . . . 370
OTP_SYS_UID_CPL_3 . . . 370
OTP_SYS_UID_CPL_4 . . . 370
OTP_SYS_UID_CPL_5 . . . 371
OTP_SYS_UID_CPL_6 . . . 371
OTP_SYS_UID_CPL_7 . . . 371
OTP_CLASS_KEY_VAL_0 . . . 371
OTP_CLASS_KEY_VAL_1 . . . 371
OTP_CLASS_KEY_VAL_2 . . . 371
OTP_CLASS_KEY_VAL_3 . . . 371
OTP_CLASS_KEY_VAL_4 . . . 372
OTP_CLASS_KEY_VAL_5 . . . 372
OTP_CLASS_KEY_VAL_6 . . . 372
OTP_CLASS_KEY_VAL_7 . . . 372
OTP_CLASS_KEY_CPL_0 . . . 372
OTP_CLASS_KEY_CPL_1 . . . 372
OTP_CLASS_KEY_CPL_2 . . . 372
OTP_CLASS_KEY_CPL_3 . . . 372
OTP_CLASS_KEY_CPL_4 . . . 373
OTP_CLASS_KEY_CPL_5 . . . 373
OTP_CLASS_KEY_CPL_6 . . . 373
OTP_CLASS_KEY_CPL_7 . . . 373
OTP_READ_LOCK . . . 373
OTP_CHANGE_STATUS . . . 373
OTP_READ_LOCK_COLD . . . 374
OTP_SEP_CHIPLET_ID_VAL_0 . . . 374
OTP_SEP_CHIPLET_ID_VAL_1 . . . 374
OTP_SEP_CHIPLET_ID_VAL_2 . . . 375
OTP_SEP_CHIPLET_ID_VAL_3 . . . 375
OTP_SEP_CHIPLET_ID_VAL_4 . . . 375
OTP_SEP_CHIPLET_ID_VAL_5 . . . 375
OTP_SEP_CHIPLET_ID_VAL_6 . . . 375
OTP_SEP_CHIPLET_ID_VAL_7 . . . 375
OTP_SEP_CHIPLET_ID_CPL_0 . . . 375

OTP_SEP_CHIPLET_ID_CPL_1 . . . 376
OTP_SEP_CHIPLET_ID_CPL_2 . . . 376
OTP_SEP_CHIPLET_ID_CPL_3 . . . 376
OTP_SEP_CHIPLET_ID_CPL_4 . . . 376
OTP_SEP_CHIPLET_ID_CPL_5 . . . 376
OTP_SEP_CHIPLET_ID_CPL_6 . . . 376
OTP_SEP_CHIPLET_ID_CPL_7 . . . 376
OTP_SEP_SIP_ID_VAL_0 . . . 376
OTP_SEP_SIP_ID_VAL_1 . . . 377
OTP_SEP_SIP_ID_VAL_2 . . . 377
OTP_SEP_SIP_ID_VAL_3 . . . 377
OTP_SEP_SIP_ID_VAL_4 . . . 377
OTP_SEP_SIP_ID_VAL_5 . . . 377
OTP_SEP_SIP_ID_VAL_6 . . . 377
OTP_SEP_SIP_ID_VAL_7 . . . 377
OTP_SEP_SIP_ID_CPL_0 . . . 378
OTP_SEP_SIP_ID_CPL_1 . . . 378
OTP_SEP_SIP_ID_CPL_2 . . . 378
OTP_SEP_SIP_ID_CPL_3 . . . 378
OTP_SEP_SIP_ID_CPL_4 . . . 378
OTP_SEP_SIP_ID_CPL_5 . . . 378
OTP_SEP_SIP_ID_CPL_6 . . . 378
OTP_SEP_SIP_ID_CPL_7 . . . 378
OTP_SEP_SYS_ID_VAL_0 . . . 379
OTP_SEP_SYS_ID_VAL_1 . . . 379
OTP_SEP_SYS_ID_VAL_2 . . . 379
OTP_SEP_SYS_ID_VAL_3 . . . 379
OTP_SEP_SYS_ID_VAL_4 . . . 379
OTP_SEP_SYS_ID_VAL_5 . . . 379
OTP_SEP_SYS_ID_VAL_6 . . . 379
OTP_SEP_SYS_ID_VAL_7 . . . 380
OTP_SEP_SYS_ID_CPL_0 . . . 380
OTP_SEP_SYS_ID_CPL_1 . . . 380
OTP_SEP_SYS_ID_CPL_2 . . . 380
OTP_SEP_SYS_ID_CPL_3 . . . 380
OTP_SEP_SYS_ID_CPL_4 . . . 380
OTP_SEP_SYS_ID_CPL_5 . . . 380
OTP_SEP_SYS_ID_CPL_6 . . . 380
OTP_SEP_SYS_ID_CPL_7 . . . 381
Key and Policy Vault Registers . . . 381
Key and Policy Vault (KM Port) . . . 381
KEY_ENTRY[64].WORD[16] . . . 381
CTRL[64] . . . 381
KPV_SCRAMBLER_KEY . . . 382

KPV_SCRAMBLER_CTRL . . . 382
DRBG Sampler Registers . . . 382
DRBG Sampler . . . 382
DATA . . . 383
CFG . . . 383
STATUS . . . 383
PREFETCH_DATA . . . 383
KM Mailbox Registers . . . 383
Key Manager Mailbox KM Side . . . 383
KM_WRITE_DATA . . . 384
KM_WRITE_SEPARATOR . . . 384
KM_READ_DATA . . . 384
KM_STATUS . . . 384
KM_IRQ_STATUS . . . 385
KM_IRQ_ENABLE . . . 385
KM_CTRL . . . 385
SEP Mailbox Registers . . . 385
Key Manager Mailbox SEP Side . . . 386
SEP_WRITE_DATA . . . 386
SEP_WRITE_SEPARATOR . . . 386
SEP_READ_DATA . . . 386
SEP_STATUS . . . 386
SEP_IRQ_STATUS . . . 387
SEP_IRQ_ENABLE . . . 387
SEP_CTRL . . . 387
Adams Bridge Key Interface Registers . . . 388
Adams Bridge Key Sideload Registers . . . 388
MLDSA_SEED.KEY_SHARE0[8] . . . 389
MLDSA_SEED.KEY_SHARE1[8] . . . 389
MLDSA_SEED.KEY_CTRL . . . 389
MLKEM_SEED_D.KEY_SHARE0[8] . . . 389
MLKEM_SEED_D.KEY_SHARE1[8] . . . 389
MLKEM_SEED_D.KEY_CTRL . . . 390
MLKEM_SEED_Z.KEY_SHARE0[8] . . . 390
MLKEM_SEED_Z.KEY_SHARE1[8] . . . 390
MLKEM_SEED_Z.KEY_CTRL . . . 390
MLKEM_MSG.KEY_SHARE0[8] . . . 390
MLKEM_MSG.KEY_SHARE1[8] . . . 390
MLKEM_MSG.KEY_CTRL . . . 390
KEY[8] . . . 391
MLKEM_SHARED_KEY.KEY_CTRL . . . 391
IRQ_STATUS . . . 391
IRQ_ENABLE . . . 391
AES Key Interface Registers . . . 391

AES Wrapper Key Registers . . . 391
KEY_SHARE0[8] . . . 392
KEY_SHARE1[8] . . . 392
KEY_CTRL . . . 392
HMAC Key Interface Registers . . . 392
HMAC Wrapper Key Registers . . . 392
KEY_SHARE0[8] . . . 393
KEY_SHARE1[8] . . . 393
KEY_CTRL . . . 393
KMAC Key Interface Registers . . . 393
KMAC Wrapper Key Registers . . . 393
KEY_SHARE0[8] . . . 393
KEY_SHARE1[8] . . . 394
KEY_CTRL . . . 394
OTBN Key Interface Registers . . . 394
OTBN Wrapper Key Registers . . . 394
KEY_SHARE0[12] . . . 394
KEY_SHARE1[12] . . . 394
KEY_CTRL . . . 395
2.1.5.12. Memory Map and Register Reference . . . 395
Memory Map . . . 395
Single-Beat Register Access . . . 396
Units Unreachable from the Inbound Port . . . 396
Detailed Register Map . . . 401
External Window Reference Model . . . 401
SEP External . . . 401
EFUSE_BANK_INIT_TIME . . . 401
2.1.6. System Management Controller (SMC) . . . 401
2.1.6.1. Overview . . . 401
Clocks and Resets . . . 403
CPU . . . 403
Boot ROM . . . 403
Interconnect . . . 403
Interrupts . . . 403
Peripherals . . . 404
Design-for-Debug and CLA . . . 404
eFuse Scan Protection . . . 404
Memory Map . . . 404
2.1.6.2. Clocks and Resets . . . 404
Clock Architecture . . . 404
Timer and Debug Timebases . . . 407
Peripheral Timing Requirements . . . 408
Reset Architecture . . . 408
Power-Good and Cold Reset . . . 409

Primary and Warm Reset . . . 409
Reset Synchronization . . . 410
Function Level Reset and Isolation . . . 410
Isolation Controls . . . 410
Timing and Recovery . . . 411
Isolation Connections . . . 411
Subsystem Controls and Clock Gating . . . 415
Memory Map and Register Reference . . . 415
Memory Map . . . 416
Detailed Register Map . . . 416
Reset Unit Address Map . . . 416
SS_CONFIG . . . 416
SS_CONFIG_LOCK . . . 416
SS_COLD_RESET_N . . . 417
SS_WARM_RESET_N . . . 417
SS_CONFIG_HOLD . . . 417
SS_SRAM_HOLD . . . 417
SS_CRITICAL_HOLD . . . 417
SS_DEBUG_HOLD . . . 417
SS_RESET_COMPLETE . . . 418
SS_COLD_RESET_LOCK . . . 418
SS_FORCE_TO_REF_CLK . . . 418
SYNC_REG . . . 418
ISOLATE_REQ_REG . . . 418
ISOLATE_REQ_PINEN_REG . . . 418
ISOLATE_REQ_SMC_REG . . . 418
ISOLATE_REQ_SMCEN_REG . . . 419
ISOLATE_REQ_VIS . . . 419
ISOLATE_REQ_FLR_COUNTER_VALUE . . . 419
ISOLATE_REQ_FLR_RESET_COUNTER_VALUE . . . 419
2.1.6.3. CPU . . . 419
Processor Core . . . 421
Local Memory . . . 421
Reset and Boot . . . 421
Memory Repair . . . 422
System Interfaces . . . 422
CPU AXI Isolation . . . 423
Drain Handshake . . . 423
Timeout Recovery . . . 423
Reset-Domain Crossing . . . 424
Interrupt Integration . . . 424
Watchdog Interface . . . 424
Bus Error Reporting . . . 424
Debug . . . 425

Debug Module Configuration . . . 425
Abstract Command Writes While cmderr Is Set . . . 426
Memory Map and Register Reference . . . 427
Memory Map . . . 427
Detailed Register Map . . . 427
CPU Control Address Map . . . 427
RESET_VECTOR[4] . . . 428
RESET_CTRL . . . 428
CORE_RESET_PULSE_COUNT . . . 429
RESET_TIMEOUT . . . 429
REFERENCE_COUNTER . . . 429
WDT_TIMEOUT . . . 429
WDT_TIMEOUT_RESET . . . 430
TEST_CTRL . . . 430
SCRATCH[16] . . . 430
WB_PC_CORE0[8] . . . 430
WB_PC_CORE1[8] . . . 430
WB_PC_CORE2[8] . . . 430
WB_PC_CORE3[8] . . . 431
SMC_ATTRIBUTES . . . 431
MUTEX[4] . . . 431
SEMA[4] . . . 431
DUMMY_ROM_0 . . . 431
DUMMY_ROM_1 . . . 432
DUMMY_ROM_2 . . . 432
DUMMY_ROM_3 . . . 432
DUMMY_ROM_NULL[4] . . . 432
CLINT Address Map . . . 432
MSIP[4] . . . 432
MTIMECMP[4] . . . 433
MTIME . . . 433
Debug Module Hart Interface . . . 433
HALTED . . . 434
GOING . . . 434
RESUMING . . . 434
EXCEPTION . . . 434
WHERETO . . . 434
ABSTRACT[5] . . . 435
PROGBUF[16] . . . 435
IMPEBREAK . . . 435
DATA[4] . . . 435
FLAGS . . . 435
ROM[26] . . . 435
Bus Error Unit Address Map . . . 435

CAUSE . . . 436
PHYS_ADDR . . . 436
ENABLE . . . 436
PLIC_ENABLE . . . 437
ACCRUED_ENABLE . . . 437
LOCAL_ENABLE . . . 437
2.1.6.4. Boot ROM . . . 438
ROM Architecture . . . 438
Hardware Configuration and Integrity . . . 438
Boot Flow . . . 438
Firmware Reference . . . 438
2.1.6.5. Interconnect . . . 438
Topology . . . 438
Fabric Topology . . . 438
Configuration Parameters . . . 441
Dual-Network Architecture . . . 442
Interfaces . . . 442
AXI Interface Widths . . . 442
Traffic Managers and Subordinates . . . 443
Traffic Managers . . . 443
Traffic Subordinates . . . 443
Address Remapping . . . 444
Alias Remapping . . . 444
Output Remapping . . . 444
Filtering and Protection . . . 445
Inbound Filtering . . . 445
Outbound Filtering . . . 447
Protection Requirements . . . 447
AXI Protection Bit Encoding . . . 447
Non-Secure Bit Filtering (AXI4 Filters) . . . 447
Full 3-Bit Protection Matching (AXI-Lite) . . . 448
Protection Bit Pass-Through . . . 448
Configuration Examples . . . 448
Routing . . . 448
Local and Remote Resource Access . . . 448
Outbound Traffic Flow . . . 449
Memory Map and Register Reference . . . 449
Memory Map . . . 449
Detailed Register Map . . . 449
Alias Remap Registers . . . 449
AXI Alias Remap Address Map . . . 449
region_start . . . 450
region_end . . . 450
region_attrs . . . 450

Output Remap Registers . . . 450
Output Remap Address Map . . . 450
region_attrs . . . 451
Inbound Traffic Filter Control Registers . . . 451
AXI Filter Control Address Map . . . 451
FILTER_CONFIG . . . 451
START_ADDR . . . 452
END_ADDR . . . 452
Outbound Traffic Filter Control Registers . . . 452
2.1.6.6. Interrupts . . . 453
Controllers . . . 453
Controller Addresses . . . 453
PLIC Sources and Contexts . . . 453
CLINT . . . 453
Bus Error Units . . . 454
Sources and Routing . . . 454
Raw Vector and PLIC Numbering . . . 454
SMC CPU Interrupt Vector Map . . . 454
Exact Indexed Map . . . 455
Routing Notes . . . 457
Software Handling . . . 457
Memory Map and Register Reference . . . 457
Memory Map . . . 457
Detailed Register Map . . . 457
PLIC Address Map . . . 457
PRIORITY[337] . . . 459
PENDING[11] . . . 459
CORE0_MEIP_ENABLE[11] . . . 459
CORE1_MEIP_ENABLE[11] . . . 459
CORE2_MEIP_ENABLE[11] . . . 459
CORE3_MEIP_ENABLE[11] . . . 459
CORE0_SEIP_ENABLE[11] . . . 459
CORE1_SEIP_ENABLE[11] . . . 460
CORE2_SEIP_ENABLE[11] . . . 460
CORE3_SEIP_ENABLE[11] . . . 460
CORE0_MEIP_THRESHOLD . . . 460
CORE0_MEIP_CLAIM_COMPLETE . . . 460
CORE1_MEIP_THRESHOLD . . . 460
CORE1_MEIP_CLAIM_COMPLETE . . . 461
CORE2_MEIP_THRESHOLD . . . 461
CORE2_MEIP_CLAIM_COMPLETE . . . 461
CORE3_MEIP_THRESHOLD . . . 461
CORE3_MEIP_CLAIM_COMPLETE . . . 461
CORE0_SEIP_THRESHOLD . . . 461

CORE0_SEIP_CLAIM_COMPLETE . . . 462
CORE1_SEIP_THRESHOLD . . . 462
CORE1_SEIP_CLAIM_COMPLETE . . . 462
CORE2_SEIP_THRESHOLD . . . 462
CORE2_SEIP_CLAIM_COMPLETE . . . 462
CORE3_SEIP_THRESHOLD . . . 463
CORE3_SEIP_CLAIM_COMPLETE . . . 463
2.1.6.7. Peripherals . . . 463
Direct Memory Access (DMA) Engine . . . 464
Architecture and Configuration . . . 464
Transfer Configuration . . . 466
Stream and Register Ownership . . . 466
Linear and Two-Dimensional Copies . . . 466
Submission and Completion . . . 467
Memory Map and Register Reference . . . 467
Memory Map . . . 467
Detailed Register Map . . . 467
iDMA Controller Address Map . . . 467
CONFIG . . . 469
STATUS_0 . . . 469
STATUS_1 . . . 470
STATUS_2 . . . 470
STATUS_3 . . . 470
STATUS_4 . . . 470
STATUS_5 . . . 470
STATUS_6 . . . 471
STATUS_7 . . . 471
STATUS_8 . . . 471
STATUS_9 . . . 471
STATUS_10 . . . 471
STATUS_11 . . . 472
STATUS_12 . . . 472
STATUS_13 . . . 472
STATUS_14 . . . 472
STATUS_15 . . . 472
NEXT_ID_0 . . . 473
NEXT_ID_1 . . . 473
NEXT_ID_2 . . . 473
NEXT_ID_3 . . . 473
NEXT_ID_4 . . . 473
NEXT_ID_5 . . . 473
NEXT_ID_6 . . . 474
NEXT_ID_7 . . . 474
NEXT_ID_8 . . . 474

NEXT_ID_9 . . . 474
NEXT_ID_10 . . . 474
NEXT_ID_11 . . . 474
NEXT_ID_12 . . . 475
NEXT_ID_13 . . . 475
NEXT_ID_14 . . . 475
NEXT_ID_15 . . . 475
DONE_0 . . . 475
DONE_1 . . . 475
DONE_2 . . . 476
DONE_3 . . . 476
DONE_4 . . . 476
DONE_5 . . . 476
DONE_6 . . . 476
DONE_7 . . . 476
DONE_8 . . . 477
DONE_9 . . . 477
DONE_10 . . . 477
DONE_11 . . . 477
DONE_12 . . . 477
DONE_13 . . . 477
DONE_14 . . . 477
DONE_15 . . . 478
DST_ADDRESS_LO . . . 478
DST_ADDRESS_HI . . . 478
SRC_ADDRESS_LO . . . 478
SRC_ADDRESS_HI . . . 478
LENGTH_LO . . . 478
LENGTH_HI . . . 478
DST_STRIDE_LO . . . 479
DST_STRIDE_HI . . . 479
SRC_STRIDE_LO . . . 479
SRC_STRIDE_HI . . . 479
NUM_REPETITIONS_LO . . . 479
NUM_REPETITIONS_HI . . . 479
Zeroer . . . 479
Use Cases . . . 480
Architecture . . . 480
State Machine . . . 480
Operation Flow . . . 481
AXI Master Interface . . . 481
Outstanding Transaction Management . . . 482
Clock Gating . . . 482
Control and Status . . . 482

Operation Control . . . 482
Status Features . . . 483
Operation Details . . . 483
Memory Map and Register Reference . . . 483
Memory Map . . . 483
Detailed Register Map . . . 483
Zeroer Control Address Map . . . 483
DEST_ADDR . . . 483
SIZE . . . 484
CTRL_STATUS . . . 484
Watchdog Timers . . . 484
Operation . . . 484
Register Access . . . 484
Memory Map and Register Reference . . . 485
Memory Map . . . 485
Detailed Register Map . . . 485
Watchdog Timer Address Map . . . 485
CTRL . . . 485
COUNT . . . 486
COUNT_HI . . . 486
SCALED_COUNT . . . 486
FEED . . . 486
KEY . . . 486
CMP . . . 487
Mailbox Unit . . . 487
Overview . . . 487
Architecture . . . 488
Paired FIFOs . . . 488
Interrupts . . . 489
Clock and Reset . . . 489
Interface and Programming . . . 489
External Interface . . . 489
Access Results . . . 489
Sending and Receiving . . . 490
Memory Map and Register Reference . . . 490
Memory Map . . . 490
Per-Mailbox Register Map . . . 493
Address Examples . . . 493
Detailed Register Map . . . 494
AXI-Lite Mailbox Address Map . . . 494
WRITE_DATA . . . 494
READ_DATA . . . 494
STATUS . . . 494
ERROR_FLAGS . . . 495

WIRQT . . . 495
RIRQT . . . 495
IRQS . . . 495
IRQEN . . . 495
IRQP . . . 496
CTRL . . . 496
Subsystem Wrapper Maps . . . 496
SMC AXI-Lite Mailbox Wrapper Address Map . . . 496
outbound_mailbox_0.WRITE_DATA . . . 515
outbound_mailbox_0.READ_DATA . . . 515
outbound_mailbox_0.STATUS . . . 515
outbound_mailbox_0.ERROR_FLAGS . . . 515
outbound_mailbox_0.WIRQT . . . 515
outbound_mailbox_0.RIRQT . . . 515
outbound_mailbox_0.IRQS . . . 516
outbound_mailbox_0.IRQEN . . . 516
outbound_mailbox_0.IRQP . . . 516
outbound_mailbox_0.CTRL . . . 516
inbound_mailbox_0.WRITE_DATA . . . 517
inbound_mailbox_0.READ_DATA . . . 517
inbound_mailbox_0.STATUS . . . 517
inbound_mailbox_0.ERROR_FLAGS . . . 517
inbound_mailbox_0.WIRQT . . . 517
inbound_mailbox_0.RIRQT . . . 517
inbound_mailbox_0.IRQS . . . 518
inbound_mailbox_0.IRQEN . . . 518
inbound_mailbox_0.IRQP . . . 518
inbound_mailbox_0.CTRL . . . 518
outbound_mailbox_1.WRITE_DATA . . . 519
outbound_mailbox_1.READ_DATA . . . 519
outbound_mailbox_1.STATUS . . . 519
outbound_mailbox_1.ERROR_FLAGS . . . 519
outbound_mailbox_1.WIRQT . . . 519
outbound_mailbox_1.RIRQT . . . 519
outbound_mailbox_1.IRQS . . . 520
outbound_mailbox_1.IRQEN . . . 520
outbound_mailbox_1.IRQP . . . 520
outbound_mailbox_1.CTRL . . . 520
inbound_mailbox_1.WRITE_DATA . . . 521
inbound_mailbox_1.READ_DATA . . . 521
inbound_mailbox_1.STATUS . . . 521
inbound_mailbox_1.ERROR_FLAGS . . . 521
inbound_mailbox_1.WIRQT . . . 521
inbound_mailbox_1.RIRQT . . . 521

inbound_mailbox_1.IRQS . . . 522
inbound_mailbox_1.IRQEN . . . 522
inbound_mailbox_1.IRQP . . . 522
inbound_mailbox_1.CTRL . . . 522
outbound_mailbox_2.WRITE_DATA . . . 523
outbound_mailbox_2.READ_DATA . . . 523
outbound_mailbox_2.STATUS . . . 523
outbound_mailbox_2.ERROR_FLAGS . . . 523
outbound_mailbox_2.WIRQT . . . 523
outbound_mailbox_2.RIRQT . . . 523
outbound_mailbox_2.IRQS . . . 524
outbound_mailbox_2.IRQEN . . . 524
outbound_mailbox_2.IRQP . . . 524
outbound_mailbox_2.CTRL . . . 524
inbound_mailbox_2.WRITE_DATA . . . 525
inbound_mailbox_2.READ_DATA . . . 525
inbound_mailbox_2.STATUS . . . 525
inbound_mailbox_2.ERROR_FLAGS . . . 525
inbound_mailbox_2.WIRQT . . . 525
inbound_mailbox_2.RIRQT . . . 525
inbound_mailbox_2.IRQS . . . 526
inbound_mailbox_2.IRQEN . . . 526
inbound_mailbox_2.IRQP . . . 526
inbound_mailbox_2.CTRL . . . 526
outbound_mailbox_3.WRITE_DATA . . . 527
outbound_mailbox_3.READ_DATA . . . 527
outbound_mailbox_3.STATUS . . . 527
outbound_mailbox_3.ERROR_FLAGS . . . 527
outbound_mailbox_3.WIRQT . . . 527
outbound_mailbox_3.RIRQT . . . 527
outbound_mailbox_3.IRQS . . . 528
outbound_mailbox_3.IRQEN . . . 528
outbound_mailbox_3.IRQP . . . 528
outbound_mailbox_3.CTRL . . . 528
inbound_mailbox_3.WRITE_DATA . . . 529
inbound_mailbox_3.READ_DATA . . . 529
inbound_mailbox_3.STATUS . . . 529
inbound_mailbox_3.ERROR_FLAGS . . . 529
inbound_mailbox_3.WIRQT . . . 529
inbound_mailbox_3.RIRQT . . . 529
inbound_mailbox_3.IRQS . . . 530
inbound_mailbox_3.IRQEN . . . 530
inbound_mailbox_3.IRQP . . . 530
inbound_mailbox_3.CTRL . . . 530

outbound_mailbox_4.WRITE_DATA . . . 531
outbound_mailbox_4.READ_DATA . . . 531
outbound_mailbox_4.STATUS . . . 531
outbound_mailbox_4.ERROR_FLAGS . . . 531
outbound_mailbox_4.WIRQT . . . 531
outbound_mailbox_4.RIRQT . . . 531
outbound_mailbox_4.IRQS . . . 532
outbound_mailbox_4.IRQEN . . . 532
outbound_mailbox_4.IRQP . . . 532
outbound_mailbox_4.CTRL . . . 532
inbound_mailbox_4.WRITE_DATA . . . 533
inbound_mailbox_4.READ_DATA . . . 533
inbound_mailbox_4.STATUS . . . 533
inbound_mailbox_4.ERROR_FLAGS . . . 533
inbound_mailbox_4.WIRQT . . . 533
inbound_mailbox_4.RIRQT . . . 533
inbound_mailbox_4.IRQS . . . 534
inbound_mailbox_4.IRQEN . . . 534
inbound_mailbox_4.IRQP . . . 534
inbound_mailbox_4.CTRL . . . 534
outbound_mailbox_5.WRITE_DATA . . . 535
outbound_mailbox_5.READ_DATA . . . 535
outbound_mailbox_5.STATUS . . . 535
outbound_mailbox_5.ERROR_FLAGS . . . 535
outbound_mailbox_5.WIRQT . . . 535
outbound_mailbox_5.RIRQT . . . 535
outbound_mailbox_5.IRQS . . . 536
outbound_mailbox_5.IRQEN . . . 536
outbound_mailbox_5.IRQP . . . 536
outbound_mailbox_5.CTRL . . . 536
inbound_mailbox_5.WRITE_DATA . . . 537
inbound_mailbox_5.READ_DATA . . . 537
inbound_mailbox_5.STATUS . . . 537
inbound_mailbox_5.ERROR_FLAGS . . . 537
inbound_mailbox_5.WIRQT . . . 537
inbound_mailbox_5.RIRQT . . . 537
inbound_mailbox_5.IRQS . . . 538
inbound_mailbox_5.IRQEN . . . 538
inbound_mailbox_5.IRQP . . . 538
inbound_mailbox_5.CTRL . . . 538
outbound_mailbox_6.WRITE_DATA . . . 539
outbound_mailbox_6.READ_DATA . . . 539
outbound_mailbox_6.STATUS . . . 539
outbound_mailbox_6.ERROR_FLAGS . . . 539

outbound_mailbox_6.WIRQT . . . 539
outbound_mailbox_6.RIRQT . . . 539
outbound_mailbox_6.IRQS . . . 540
outbound_mailbox_6.IRQEN . . . 540
outbound_mailbox_6.IRQP . . . 540
outbound_mailbox_6.CTRL . . . 540
inbound_mailbox_6.WRITE_DATA . . . 541
inbound_mailbox_6.READ_DATA . . . 541
inbound_mailbox_6.STATUS . . . 541
inbound_mailbox_6.ERROR_FLAGS . . . 541
inbound_mailbox_6.WIRQT . . . 541
inbound_mailbox_6.RIRQT . . . 541
inbound_mailbox_6.IRQS . . . 542
inbound_mailbox_6.IRQEN . . . 542
inbound_mailbox_6.IRQP . . . 542
inbound_mailbox_6.CTRL . . . 542
outbound_mailbox_7.WRITE_DATA . . . 543
outbound_mailbox_7.READ_DATA . . . 543
outbound_mailbox_7.STATUS . . . 543
outbound_mailbox_7.ERROR_FLAGS . . . 543
outbound_mailbox_7.WIRQT . . . 543
outbound_mailbox_7.RIRQT . . . 543
outbound_mailbox_7.IRQS . . . 544
outbound_mailbox_7.IRQEN . . . 544
outbound_mailbox_7.IRQP . . . 544
outbound_mailbox_7.CTRL . . . 544
inbound_mailbox_7.WRITE_DATA . . . 545
inbound_mailbox_7.READ_DATA . . . 545
inbound_mailbox_7.STATUS . . . 545
inbound_mailbox_7.ERROR_FLAGS . . . 545
inbound_mailbox_7.WIRQT . . . 545
inbound_mailbox_7.RIRQT . . . 545
inbound_mailbox_7.IRQS . . . 546
inbound_mailbox_7.IRQEN . . . 546
inbound_mailbox_7.IRQP . . . 546
inbound_mailbox_7.CTRL . . . 546
outbound_mailbox_8.WRITE_DATA . . . 547
outbound_mailbox_8.READ_DATA . . . 547
outbound_mailbox_8.STATUS . . . 547
outbound_mailbox_8.ERROR_FLAGS . . . 547
outbound_mailbox_8.WIRQT . . . 547
outbound_mailbox_8.RIRQT . . . 547
outbound_mailbox_8.IRQS . . . 548
outbound_mailbox_8.IRQEN . . . 548

outbound_mailbox_8.IRQP . . . 548
outbound_mailbox_8.CTRL . . . 548
inbound_mailbox_8.WRITE_DATA . . . 549
inbound_mailbox_8.READ_DATA . . . 549
inbound_mailbox_8.STATUS . . . 549
inbound_mailbox_8.ERROR_FLAGS . . . 549
inbound_mailbox_8.WIRQT . . . 549
inbound_mailbox_8.RIRQT . . . 549
inbound_mailbox_8.IRQS . . . 550
inbound_mailbox_8.IRQEN . . . 550
inbound_mailbox_8.IRQP . . . 550
inbound_mailbox_8.CTRL . . . 550
outbound_mailbox_9.WRITE_DATA . . . 551
outbound_mailbox_9.READ_DATA . . . 551
outbound_mailbox_9.STATUS . . . 551
outbound_mailbox_9.ERROR_FLAGS . . . 551
outbound_mailbox_9.WIRQT . . . 551
outbound_mailbox_9.RIRQT . . . 551
outbound_mailbox_9.IRQS . . . 552
outbound_mailbox_9.IRQEN . . . 552
outbound_mailbox_9.IRQP . . . 552
outbound_mailbox_9.CTRL . . . 552
inbound_mailbox_9.WRITE_DATA . . . 553
inbound_mailbox_9.READ_DATA . . . 553
inbound_mailbox_9.STATUS . . . 553
inbound_mailbox_9.ERROR_FLAGS . . . 553
inbound_mailbox_9.WIRQT . . . 553
inbound_mailbox_9.RIRQT . . . 553
inbound_mailbox_9.IRQS . . . 554
inbound_mailbox_9.IRQEN . . . 554
inbound_mailbox_9.IRQP . . . 554
inbound_mailbox_9.CTRL . . . 554
outbound_mailbox_10.WRITE_DATA . . . 555
outbound_mailbox_10.READ_DATA . . . 555
outbound_mailbox_10.STATUS . . . 555
outbound_mailbox_10.ERROR_FLAGS . . . 555
outbound_mailbox_10.WIRQT . . . 555
outbound_mailbox_10.RIRQT . . . 555
outbound_mailbox_10.IRQS . . . 556
outbound_mailbox_10.IRQEN . . . 556
outbound_mailbox_10.IRQP . . . 556
outbound_mailbox_10.CTRL . . . 556
inbound_mailbox_10.WRITE_DATA . . . 557
inbound_mailbox_10.READ_DATA . . . 557

inbound_mailbox_10.STATUS . . . 557
inbound_mailbox_10.ERROR_FLAGS . . . 557
inbound_mailbox_10.WIRQT . . . 557
inbound_mailbox_10.RIRQT . . . 557
inbound_mailbox_10.IRQS . . . 558
inbound_mailbox_10.IRQEN . . . 558
inbound_mailbox_10.IRQP . . . 558
inbound_mailbox_10.CTRL . . . 558
outbound_mailbox_11.WRITE_DATA . . . 559
outbound_mailbox_11.READ_DATA . . . 559
outbound_mailbox_11.STATUS . . . 559
outbound_mailbox_11.ERROR_FLAGS . . . 559
outbound_mailbox_11.WIRQT . . . 559
outbound_mailbox_11.RIRQT . . . 559
outbound_mailbox_11.IRQS . . . 560
outbound_mailbox_11.IRQEN . . . 560
outbound_mailbox_11.IRQP . . . 560
outbound_mailbox_11.CTRL . . . 560
inbound_mailbox_11.WRITE_DATA . . . 561
inbound_mailbox_11.READ_DATA . . . 561
inbound_mailbox_11.STATUS . . . 561
inbound_mailbox_11.ERROR_FLAGS . . . 561
inbound_mailbox_11.WIRQT . . . 561
inbound_mailbox_11.RIRQT . . . 561
inbound_mailbox_11.IRQS . . . 562
inbound_mailbox_11.IRQEN . . . 562
inbound_mailbox_11.IRQP . . . 562
inbound_mailbox_11.CTRL . . . 562
outbound_mailbox_12.WRITE_DATA . . . 563
outbound_mailbox_12.READ_DATA . . . 563
outbound_mailbox_12.STATUS . . . 563
outbound_mailbox_12.ERROR_FLAGS . . . 563
outbound_mailbox_12.WIRQT . . . 563
outbound_mailbox_12.RIRQT . . . 563
outbound_mailbox_12.IRQS . . . 564
outbound_mailbox_12.IRQEN . . . 564
outbound_mailbox_12.IRQP . . . 564
outbound_mailbox_12.CTRL . . . 564
inbound_mailbox_12.WRITE_DATA . . . 565
inbound_mailbox_12.READ_DATA . . . 565
inbound_mailbox_12.STATUS . . . 565
inbound_mailbox_12.ERROR_FLAGS . . . 565
inbound_mailbox_12.WIRQT . . . 565
inbound_mailbox_12.RIRQT . . . 565

inbound_mailbox_12.IRQS . . . 566
inbound_mailbox_12.IRQEN . . . 566
inbound_mailbox_12.IRQP . . . 566
inbound_mailbox_12.CTRL . . . 566
outbound_mailbox_13.WRITE_DATA . . . 567
outbound_mailbox_13.READ_DATA . . . 567
outbound_mailbox_13.STATUS . . . 567
outbound_mailbox_13.ERROR_FLAGS . . . 567
outbound_mailbox_13.WIRQT . . . 567
outbound_mailbox_13.RIRQT . . . 567
outbound_mailbox_13.IRQS . . . 568
outbound_mailbox_13.IRQEN . . . 568
outbound_mailbox_13.IRQP . . . 568
outbound_mailbox_13.CTRL . . . 568
inbound_mailbox_13.WRITE_DATA . . . 569
inbound_mailbox_13.READ_DATA . . . 569
inbound_mailbox_13.STATUS . . . 569
inbound_mailbox_13.ERROR_FLAGS . . . 569
inbound_mailbox_13.WIRQT . . . 569
inbound_mailbox_13.RIRQT . . . 569
inbound_mailbox_13.IRQS . . . 570
inbound_mailbox_13.IRQEN . . . 570
inbound_mailbox_13.IRQP . . . 570
inbound_mailbox_13.CTRL . . . 570
outbound_mailbox_14.WRITE_DATA . . . 571
outbound_mailbox_14.READ_DATA . . . 571
outbound_mailbox_14.STATUS . . . 571
outbound_mailbox_14.ERROR_FLAGS . . . 571
outbound_mailbox_14.WIRQT . . . 571
outbound_mailbox_14.RIRQT . . . 571
outbound_mailbox_14.IRQS . . . 572
outbound_mailbox_14.IRQEN . . . 572
outbound_mailbox_14.IRQP . . . 572
outbound_mailbox_14.CTRL . . . 572
inbound_mailbox_14.WRITE_DATA . . . 573
inbound_mailbox_14.READ_DATA . . . 573
inbound_mailbox_14.STATUS . . . 573
inbound_mailbox_14.ERROR_FLAGS . . . 573
inbound_mailbox_14.WIRQT . . . 573
inbound_mailbox_14.RIRQT . . . 573
inbound_mailbox_14.IRQS . . . 574
inbound_mailbox_14.IRQEN . . . 574
inbound_mailbox_14.IRQP . . . 574
inbound_mailbox_14.CTRL . . . 574

outbound_mailbox_15.WRITE_DATA . . . 575
outbound_mailbox_15.READ_DATA . . . 575
outbound_mailbox_15.STATUS . . . 575
outbound_mailbox_15.ERROR_FLAGS . . . 575
outbound_mailbox_15.WIRQT . . . 575
outbound_mailbox_15.RIRQT . . . 575
outbound_mailbox_15.IRQS . . . 576
outbound_mailbox_15.IRQEN . . . 576
outbound_mailbox_15.IRQP . . . 576
outbound_mailbox_15.CTRL . . . 576
inbound_mailbox_15.WRITE_DATA . . . 577
inbound_mailbox_15.READ_DATA . . . 577
inbound_mailbox_15.STATUS . . . 577
inbound_mailbox_15.ERROR_FLAGS . . . 577
inbound_mailbox_15.WIRQT . . . 577
inbound_mailbox_15.RIRQT . . . 577
inbound_mailbox_15.IRQS . . . 578
inbound_mailbox_15.IRQEN . . . 578
inbound_mailbox_15.IRQP . . . 578
inbound_mailbox_15.CTRL . . . 578
outbound_mailbox_16.WRITE_DATA . . . 579
outbound_mailbox_16.READ_DATA . . . 579
outbound_mailbox_16.STATUS . . . 579
outbound_mailbox_16.ERROR_FLAGS . . . 579
outbound_mailbox_16.WIRQT . . . 579
outbound_mailbox_16.RIRQT . . . 579
outbound_mailbox_16.IRQS . . . 580
outbound_mailbox_16.IRQEN . . . 580
outbound_mailbox_16.IRQP . . . 580
outbound_mailbox_16.CTRL . . . 580
inbound_mailbox_16.WRITE_DATA . . . 581
inbound_mailbox_16.READ_DATA . . . 581
inbound_mailbox_16.STATUS . . . 581
inbound_mailbox_16.ERROR_FLAGS . . . 581
inbound_mailbox_16.WIRQT . . . 581
inbound_mailbox_16.RIRQT . . . 581
inbound_mailbox_16.IRQS . . . 582
inbound_mailbox_16.IRQEN . . . 582
inbound_mailbox_16.IRQP . . . 582
inbound_mailbox_16.CTRL . . . 582
outbound_mailbox_17.WRITE_DATA . . . 583
outbound_mailbox_17.READ_DATA . . . 583
outbound_mailbox_17.STATUS . . . 583
outbound_mailbox_17.ERROR_FLAGS . . . 583

outbound_mailbox_17.WIRQT . . . 583
outbound_mailbox_17.RIRQT . . . 583
outbound_mailbox_17.IRQS . . . 584
outbound_mailbox_17.IRQEN . . . 584
outbound_mailbox_17.IRQP . . . 584
outbound_mailbox_17.CTRL . . . 584
inbound_mailbox_17.WRITE_DATA . . . 585
inbound_mailbox_17.READ_DATA . . . 585
inbound_mailbox_17.STATUS . . . 585
inbound_mailbox_17.ERROR_FLAGS . . . 585
inbound_mailbox_17.WIRQT . . . 585
inbound_mailbox_17.RIRQT . . . 585
inbound_mailbox_17.IRQS . . . 586
inbound_mailbox_17.IRQEN . . . 586
inbound_mailbox_17.IRQP . . . 586
inbound_mailbox_17.CTRL . . . 586
outbound_mailbox_18.WRITE_DATA . . . 587
outbound_mailbox_18.READ_DATA . . . 587
outbound_mailbox_18.STATUS . . . 587
outbound_mailbox_18.ERROR_FLAGS . . . 587
outbound_mailbox_18.WIRQT . . . 587
outbound_mailbox_18.RIRQT . . . 587
outbound_mailbox_18.IRQS . . . 588
outbound_mailbox_18.IRQEN . . . 588
outbound_mailbox_18.IRQP . . . 588
outbound_mailbox_18.CTRL . . . 588
inbound_mailbox_18.WRITE_DATA . . . 589
inbound_mailbox_18.READ_DATA . . . 589
inbound_mailbox_18.STATUS . . . 589
inbound_mailbox_18.ERROR_FLAGS . . . 589
inbound_mailbox_18.WIRQT . . . 589
inbound_mailbox_18.RIRQT . . . 589
inbound_mailbox_18.IRQS . . . 590
inbound_mailbox_18.IRQEN . . . 590
inbound_mailbox_18.IRQP . . . 590
inbound_mailbox_18.CTRL . . . 590
outbound_mailbox_19.WRITE_DATA . . . 591
outbound_mailbox_19.READ_DATA . . . 591
outbound_mailbox_19.STATUS . . . 591
outbound_mailbox_19.ERROR_FLAGS . . . 591
outbound_mailbox_19.WIRQT . . . 591
outbound_mailbox_19.RIRQT . . . 591
outbound_mailbox_19.IRQS . . . 592
outbound_mailbox_19.IRQEN . . . 592

outbound_mailbox_19.IRQP . . . 592
outbound_mailbox_19.CTRL . . . 592
inbound_mailbox_19.WRITE_DATA . . . 593
inbound_mailbox_19.READ_DATA . . . 593
inbound_mailbox_19.STATUS . . . 593
inbound_mailbox_19.ERROR_FLAGS . . . 593
inbound_mailbox_19.WIRQT . . . 593
inbound_mailbox_19.RIRQT . . . 593
inbound_mailbox_19.IRQS . . . 594
inbound_mailbox_19.IRQEN . . . 594
inbound_mailbox_19.IRQP . . . 594
inbound_mailbox_19.CTRL . . . 594
outbound_mailbox_20.WRITE_DATA . . . 595
outbound_mailbox_20.READ_DATA . . . 595
outbound_mailbox_20.STATUS . . . 595
outbound_mailbox_20.ERROR_FLAGS . . . 595
outbound_mailbox_20.WIRQT . . . 595
outbound_mailbox_20.RIRQT . . . 595
outbound_mailbox_20.IRQS . . . 596
outbound_mailbox_20.IRQEN . . . 596
outbound_mailbox_20.IRQP . . . 596
outbound_mailbox_20.CTRL . . . 596
inbound_mailbox_20.WRITE_DATA . . . 597
inbound_mailbox_20.READ_DATA . . . 597
inbound_mailbox_20.STATUS . . . 597
inbound_mailbox_20.ERROR_FLAGS . . . 597
inbound_mailbox_20.WIRQT . . . 597
inbound_mailbox_20.RIRQT . . . 597
inbound_mailbox_20.IRQS . . . 598
inbound_mailbox_20.IRQEN . . . 598
inbound_mailbox_20.IRQP . . . 598
inbound_mailbox_20.CTRL . . . 598
outbound_mailbox_21.WRITE_DATA . . . 599
outbound_mailbox_21.READ_DATA . . . 599
outbound_mailbox_21.STATUS . . . 599
outbound_mailbox_21.ERROR_FLAGS . . . 599
outbound_mailbox_21.WIRQT . . . 599
outbound_mailbox_21.RIRQT . . . 599
outbound_mailbox_21.IRQS . . . 600
outbound_mailbox_21.IRQEN . . . 600
outbound_mailbox_21.IRQP . . . 600
outbound_mailbox_21.CTRL . . . 600
inbound_mailbox_21.WRITE_DATA . . . 601
inbound_mailbox_21.READ_DATA . . . 601

inbound_mailbox_21.STATUS . . . 601
inbound_mailbox_21.ERROR_FLAGS . . . 601
inbound_mailbox_21.WIRQT . . . 601
inbound_mailbox_21.RIRQT . . . 601
inbound_mailbox_21.IRQS . . . 602
inbound_mailbox_21.IRQEN . . . 602
inbound_mailbox_21.IRQP . . . 602
inbound_mailbox_21.CTRL . . . 602
outbound_mailbox_22.WRITE_DATA . . . 603
outbound_mailbox_22.READ_DATA . . . 603
outbound_mailbox_22.STATUS . . . 603
outbound_mailbox_22.ERROR_FLAGS . . . 603
outbound_mailbox_22.WIRQT . . . 603
outbound_mailbox_22.RIRQT . . . 603
outbound_mailbox_22.IRQS . . . 604
outbound_mailbox_22.IRQEN . . . 604
outbound_mailbox_22.IRQP . . . 604
outbound_mailbox_22.CTRL . . . 604
inbound_mailbox_22.WRITE_DATA . . . 605
inbound_mailbox_22.READ_DATA . . . 605
inbound_mailbox_22.STATUS . . . 605
inbound_mailbox_22.ERROR_FLAGS . . . 605
inbound_mailbox_22.WIRQT . . . 605
inbound_mailbox_22.RIRQT . . . 605
inbound_mailbox_22.IRQS . . . 606
inbound_mailbox_22.IRQEN . . . 606
inbound_mailbox_22.IRQP . . . 606
inbound_mailbox_22.CTRL . . . 606
outbound_mailbox_23.WRITE_DATA . . . 607
outbound_mailbox_23.READ_DATA . . . 607
outbound_mailbox_23.STATUS . . . 607
outbound_mailbox_23.ERROR_FLAGS . . . 607
outbound_mailbox_23.WIRQT . . . 607
outbound_mailbox_23.RIRQT . . . 607
outbound_mailbox_23.IRQS . . . 608
outbound_mailbox_23.IRQEN . . . 608
outbound_mailbox_23.IRQP . . . 608
outbound_mailbox_23.CTRL . . . 608
inbound_mailbox_23.WRITE_DATA . . . 609
inbound_mailbox_23.READ_DATA . . . 609
inbound_mailbox_23.STATUS . . . 609
inbound_mailbox_23.ERROR_FLAGS . . . 609
inbound_mailbox_23.WIRQT . . . 609
inbound_mailbox_23.RIRQT . . . 609

inbound_mailbox_23.IRQS . . . 610
inbound_mailbox_23.IRQEN . . . 610
inbound_mailbox_23.IRQP . . . 610
inbound_mailbox_23.CTRL . . . 610
outbound_mailbox_24.WRITE_DATA . . . 611
outbound_mailbox_24.READ_DATA . . . 611
outbound_mailbox_24.STATUS . . . 611
outbound_mailbox_24.ERROR_FLAGS . . . 611
outbound_mailbox_24.WIRQT . . . 611
outbound_mailbox_24.RIRQT . . . 611
outbound_mailbox_24.IRQS . . . 612
outbound_mailbox_24.IRQEN . . . 612
outbound_mailbox_24.IRQP . . . 612
outbound_mailbox_24.CTRL . . . 612
inbound_mailbox_24.WRITE_DATA . . . 613
inbound_mailbox_24.READ_DATA . . . 613
inbound_mailbox_24.STATUS . . . 613
inbound_mailbox_24.ERROR_FLAGS . . . 613
inbound_mailbox_24.WIRQT . . . 613
inbound_mailbox_24.RIRQT . . . 613
inbound_mailbox_24.IRQS . . . 614
inbound_mailbox_24.IRQEN . . . 614
inbound_mailbox_24.IRQP . . . 614
inbound_mailbox_24.CTRL . . . 614
outbound_mailbox_25.WRITE_DATA . . . 615
outbound_mailbox_25.READ_DATA . . . 615
outbound_mailbox_25.STATUS . . . 615
outbound_mailbox_25.ERROR_FLAGS . . . 615
outbound_mailbox_25.WIRQT . . . 615
outbound_mailbox_25.RIRQT . . . 615
outbound_mailbox_25.IRQS . . . 616
outbound_mailbox_25.IRQEN . . . 616
outbound_mailbox_25.IRQP . . . 616
outbound_mailbox_25.CTRL . . . 616
inbound_mailbox_25.WRITE_DATA . . . 617
inbound_mailbox_25.READ_DATA . . . 617
inbound_mailbox_25.STATUS . . . 617
inbound_mailbox_25.ERROR_FLAGS . . . 617
inbound_mailbox_25.WIRQT . . . 617
inbound_mailbox_25.RIRQT . . . 617
inbound_mailbox_25.IRQS . . . 618
inbound_mailbox_25.IRQEN . . . 618
inbound_mailbox_25.IRQP . . . 618
inbound_mailbox_25.CTRL . . . 618

outbound_mailbox_26.WRITE_DATA . . . 619
outbound_mailbox_26.READ_DATA . . . 619
outbound_mailbox_26.STATUS . . . 619
outbound_mailbox_26.ERROR_FLAGS . . . 619
outbound_mailbox_26.WIRQT . . . 619
outbound_mailbox_26.RIRQT . . . 619
outbound_mailbox_26.IRQS . . . 620
outbound_mailbox_26.IRQEN . . . 620
outbound_mailbox_26.IRQP . . . 620
outbound_mailbox_26.CTRL . . . 620
inbound_mailbox_26.WRITE_DATA . . . 621
inbound_mailbox_26.READ_DATA . . . 621
inbound_mailbox_26.STATUS . . . 621
inbound_mailbox_26.ERROR_FLAGS . . . 621
inbound_mailbox_26.WIRQT . . . 621
inbound_mailbox_26.RIRQT . . . 621
inbound_mailbox_26.IRQS . . . 622
inbound_mailbox_26.IRQEN . . . 622
inbound_mailbox_26.IRQP . . . 622
inbound_mailbox_26.CTRL . . . 622
outbound_mailbox_27.WRITE_DATA . . . 623
outbound_mailbox_27.READ_DATA . . . 623
outbound_mailbox_27.STATUS . . . 623
outbound_mailbox_27.ERROR_FLAGS . . . 623
outbound_mailbox_27.WIRQT . . . 623
outbound_mailbox_27.RIRQT . . . 623
outbound_mailbox_27.IRQS . . . 624
outbound_mailbox_27.IRQEN . . . 624
outbound_mailbox_27.IRQP . . . 624
outbound_mailbox_27.CTRL . . . 624
inbound_mailbox_27.WRITE_DATA . . . 625
inbound_mailbox_27.READ_DATA . . . 625
inbound_mailbox_27.STATUS . . . 625
inbound_mailbox_27.ERROR_FLAGS . . . 625
inbound_mailbox_27.WIRQT . . . 625
inbound_mailbox_27.RIRQT . . . 625
inbound_mailbox_27.IRQS . . . 626
inbound_mailbox_27.IRQEN . . . 626
inbound_mailbox_27.IRQP . . . 626
inbound_mailbox_27.CTRL . . . 626
outbound_mailbox_28.WRITE_DATA . . . 627
outbound_mailbox_28.READ_DATA . . . 627
outbound_mailbox_28.STATUS . . . 627
outbound_mailbox_28.ERROR_FLAGS . . . 627

outbound_mailbox_28.WIRQT . . . 627
outbound_mailbox_28.RIRQT . . . 627
outbound_mailbox_28.IRQS . . . 628
outbound_mailbox_28.IRQEN . . . 628
outbound_mailbox_28.IRQP . . . 628
outbound_mailbox_28.CTRL . . . 628
inbound_mailbox_28.WRITE_DATA . . . 629
inbound_mailbox_28.READ_DATA . . . 629
inbound_mailbox_28.STATUS . . . 629
inbound_mailbox_28.ERROR_FLAGS . . . 629
inbound_mailbox_28.WIRQT . . . 629
inbound_mailbox_28.RIRQT . . . 629
inbound_mailbox_28.IRQS . . . 630
inbound_mailbox_28.IRQEN . . . 630
inbound_mailbox_28.IRQP . . . 630
inbound_mailbox_28.CTRL . . . 630
outbound_mailbox_29.WRITE_DATA . . . 631
outbound_mailbox_29.READ_DATA . . . 631
outbound_mailbox_29.STATUS . . . 631
outbound_mailbox_29.ERROR_FLAGS . . . 631
outbound_mailbox_29.WIRQT . . . 631
outbound_mailbox_29.RIRQT . . . 631
outbound_mailbox_29.IRQS . . . 632
outbound_mailbox_29.IRQEN . . . 632
outbound_mailbox_29.IRQP . . . 632
outbound_mailbox_29.CTRL . . . 632
inbound_mailbox_29.WRITE_DATA . . . 633
inbound_mailbox_29.READ_DATA . . . 633
inbound_mailbox_29.STATUS . . . 633
inbound_mailbox_29.ERROR_FLAGS . . . 633
inbound_mailbox_29.WIRQT . . . 633
inbound_mailbox_29.RIRQT . . . 633
inbound_mailbox_29.IRQS . . . 634
inbound_mailbox_29.IRQEN . . . 634
inbound_mailbox_29.IRQP . . . 634
inbound_mailbox_29.CTRL . . . 634
outbound_mailbox_30.WRITE_DATA . . . 635
outbound_mailbox_30.READ_DATA . . . 635
outbound_mailbox_30.STATUS . . . 635
outbound_mailbox_30.ERROR_FLAGS . . . 635
outbound_mailbox_30.WIRQT . . . 635
outbound_mailbox_30.RIRQT . . . 635
outbound_mailbox_30.IRQS . . . 636
outbound_mailbox_30.IRQEN . . . 636

outbound_mailbox_30.IRQP . . . 636
outbound_mailbox_30.CTRL . . . 636
inbound_mailbox_30.WRITE_DATA . . . 637
inbound_mailbox_30.READ_DATA . . . 637
inbound_mailbox_30.STATUS . . . 637
inbound_mailbox_30.ERROR_FLAGS . . . 637
inbound_mailbox_30.WIRQT . . . 637
inbound_mailbox_30.RIRQT . . . 637
inbound_mailbox_30.IRQS . . . 638
inbound_mailbox_30.IRQEN . . . 638
inbound_mailbox_30.IRQP . . . 638
inbound_mailbox_30.CTRL . . . 638
outbound_mailbox_31.WRITE_DATA . . . 639
outbound_mailbox_31.READ_DATA . . . 639
outbound_mailbox_31.STATUS . . . 639
outbound_mailbox_31.ERROR_FLAGS . . . 639
outbound_mailbox_31.WIRQT . . . 639
outbound_mailbox_31.RIRQT . . . 639
outbound_mailbox_31.IRQS . . . 640
outbound_mailbox_31.IRQEN . . . 640
outbound_mailbox_31.IRQP . . . 640
outbound_mailbox_31.CTRL . . . 640
inbound_mailbox_31.WRITE_DATA . . . 641
inbound_mailbox_31.READ_DATA . . . 641
inbound_mailbox_31.STATUS . . . 641
inbound_mailbox_31.ERROR_FLAGS . . . 641
inbound_mailbox_31.WIRQT . . . 641
inbound_mailbox_31.RIRQT . . . 641
inbound_mailbox_31.IRQS . . . 642
inbound_mailbox_31.IRQEN . . . 642
inbound_mailbox_31.IRQP . . . 642
inbound_mailbox_31.CTRL . . . 642
SEP AXI-Lite Mailbox Wrapper Address Map . . . 643
outbound_mailbox_0.WRITE_DATA . . . 647
outbound_mailbox_0.READ_DATA . . . 647
outbound_mailbox_0.STATUS . . . 648
outbound_mailbox_0.ERROR_FLAGS . . . 648
outbound_mailbox_0.WIRQT . . . 648
outbound_mailbox_0.RIRQT . . . 648
outbound_mailbox_0.IRQS . . . 648
outbound_mailbox_0.IRQEN . . . 649
outbound_mailbox_0.IRQP . . . 649
outbound_mailbox_0.CTRL . . . 649
inbound_mailbox_0.WRITE_DATA . . . 649

inbound_mailbox_0.READ_DATA . . . 649
inbound_mailbox_0.STATUS . . . 650
inbound_mailbox_0.ERROR_FLAGS . . . 650
inbound_mailbox_0.WIRQT . . . 650
inbound_mailbox_0.RIRQT . . . 650
inbound_mailbox_0.IRQS . . . 650
inbound_mailbox_0.IRQEN . . . 651
inbound_mailbox_0.IRQP . . . 651
inbound_mailbox_0.CTRL . . . 651
outbound_mailbox_1.WRITE_DATA . . . 651
outbound_mailbox_1.READ_DATA . . . 651
outbound_mailbox_1.STATUS . . . 651
outbound_mailbox_1.ERROR_FLAGS . . . 652
outbound_mailbox_1.WIRQT . . . 652
outbound_mailbox_1.RIRQT . . . 652
outbound_mailbox_1.IRQS . . . 652
outbound_mailbox_1.IRQEN . . . 653
outbound_mailbox_1.IRQP . . . 653
outbound_mailbox_1.CTRL . . . 653
inbound_mailbox_1.WRITE_DATA . . . 653
inbound_mailbox_1.READ_DATA . . . 653
inbound_mailbox_1.STATUS . . . 653
inbound_mailbox_1.ERROR_FLAGS . . . 654
inbound_mailbox_1.WIRQT . . . 654
inbound_mailbox_1.RIRQT . . . 654
inbound_mailbox_1.IRQS . . . 654
inbound_mailbox_1.IRQEN . . . 654
inbound_mailbox_1.IRQP . . . 655
inbound_mailbox_1.CTRL . . . 655
outbound_mailbox_2.WRITE_DATA . . . 655
outbound_mailbox_2.READ_DATA . . . 655
outbound_mailbox_2.STATUS . . . 655
outbound_mailbox_2.ERROR_FLAGS . . . 656
outbound_mailbox_2.WIRQT . . . 656
outbound_mailbox_2.RIRQT . . . 656
outbound_mailbox_2.IRQS . . . 656
outbound_mailbox_2.IRQEN . . . 656
outbound_mailbox_2.IRQP . . . 657
outbound_mailbox_2.CTRL . . . 657
inbound_mailbox_2.WRITE_DATA . . . 657
inbound_mailbox_2.READ_DATA . . . 657
inbound_mailbox_2.STATUS . . . 657
inbound_mailbox_2.ERROR_FLAGS . . . 657
inbound_mailbox_2.WIRQT . . . 658

inbound_mailbox_2.RIRQT . . . 658
inbound_mailbox_2.IRQS . . . 658
inbound_mailbox_2.IRQEN . . . 658
inbound_mailbox_2.IRQP . . . 659
inbound_mailbox_2.CTRL . . . 659
outbound_mailbox_3.WRITE_DATA . . . 659
outbound_mailbox_3.READ_DATA . . . 659
outbound_mailbox_3.STATUS . . . 659
outbound_mailbox_3.ERROR_FLAGS . . . 659
outbound_mailbox_3.WIRQT . . . 660
outbound_mailbox_3.RIRQT . . . 660
outbound_mailbox_3.IRQS . . . 660
outbound_mailbox_3.IRQEN . . . 660
outbound_mailbox_3.IRQP . . . 660
outbound_mailbox_3.CTRL . . . 661
inbound_mailbox_3.WRITE_DATA . . . 661
inbound_mailbox_3.READ_DATA . . . 661
inbound_mailbox_3.STATUS . . . 661
inbound_mailbox_3.ERROR_FLAGS . . . 661
inbound_mailbox_3.WIRQT . . . 662
inbound_mailbox_3.RIRQT . . . 662
inbound_mailbox_3.IRQS . . . 662
inbound_mailbox_3.IRQEN . . . 662
inbound_mailbox_3.IRQP . . . 662
inbound_mailbox_3.CTRL . . . 663
outbound_mailbox_4.WRITE_DATA . . . 663
outbound_mailbox_4.READ_DATA . . . 663
outbound_mailbox_4.STATUS . . . 663
outbound_mailbox_4.ERROR_FLAGS . . . 663
outbound_mailbox_4.WIRQT . . . 663
outbound_mailbox_4.RIRQT . . . 664
outbound_mailbox_4.IRQS . . . 664
outbound_mailbox_4.IRQEN . . . 664
outbound_mailbox_4.IRQP . . . 664
outbound_mailbox_4.CTRL . . . 665
inbound_mailbox_4.WRITE_DATA . . . 665
inbound_mailbox_4.READ_DATA . . . 665
inbound_mailbox_4.STATUS . . . 665
inbound_mailbox_4.ERROR_FLAGS . . . 665
inbound_mailbox_4.WIRQT . . . 665
inbound_mailbox_4.RIRQT . . . 666
inbound_mailbox_4.IRQS . . . 666
inbound_mailbox_4.IRQEN . . . 666
inbound_mailbox_4.IRQP . . . 666

inbound_mailbox_4.CTRL . . . 667
outbound_mailbox_5.WRITE_DATA . . . 667
outbound_mailbox_5.READ_DATA . . . 667
outbound_mailbox_5.STATUS . . . 667
outbound_mailbox_5.ERROR_FLAGS . . . 667
outbound_mailbox_5.WIRQT . . . 667
outbound_mailbox_5.RIRQT . . . 668
outbound_mailbox_5.IRQS . . . 668
outbound_mailbox_5.IRQEN . . . 668
outbound_mailbox_5.IRQP . . . 668
outbound_mailbox_5.CTRL . . . 669
inbound_mailbox_5.WRITE_DATA . . . 669
inbound_mailbox_5.READ_DATA . . . 669
inbound_mailbox_5.STATUS . . . 669
inbound_mailbox_5.ERROR_FLAGS . . . 669
inbound_mailbox_5.WIRQT . . . 669
inbound_mailbox_5.RIRQT . . . 670
inbound_mailbox_5.IRQS . . . 670
inbound_mailbox_5.IRQEN . . . 670
inbound_mailbox_5.IRQP . . . 670
inbound_mailbox_5.CTRL . . . 671
outbound_mailbox_6.WRITE_DATA . . . 671
outbound_mailbox_6.READ_DATA . . . 671
outbound_mailbox_6.STATUS . . . 671
outbound_mailbox_6.ERROR_FLAGS . . . 671
outbound_mailbox_6.WIRQT . . . 671
outbound_mailbox_6.RIRQT . . . 672
outbound_mailbox_6.IRQS . . . 672
outbound_mailbox_6.IRQEN . . . 672
outbound_mailbox_6.IRQP . . . 672
outbound_mailbox_6.CTRL . . . 673
inbound_mailbox_6.WRITE_DATA . . . 673
inbound_mailbox_6.READ_DATA . . . 673
inbound_mailbox_6.STATUS . . . 673
inbound_mailbox_6.ERROR_FLAGS . . . 673
inbound_mailbox_6.WIRQT . . . 673
inbound_mailbox_6.RIRQT . . . 674
inbound_mailbox_6.IRQS . . . 674
inbound_mailbox_6.IRQEN . . . 674
inbound_mailbox_6.IRQP . . . 674
inbound_mailbox_6.CTRL . . . 675
outbound_mailbox_7.WRITE_DATA . . . 675
outbound_mailbox_7.READ_DATA . . . 675
outbound_mailbox_7.STATUS . . . 675

outbound_mailbox_7.ERROR_FLAGS . . . 675
outbound_mailbox_7.WIRQT . . . 675
outbound_mailbox_7.RIRQT . . . 676
outbound_mailbox_7.IRQS . . . 676
outbound_mailbox_7.IRQEN . . . 676
outbound_mailbox_7.IRQP . . . 676
outbound_mailbox_7.CTRL . . . 677
inbound_mailbox_7.WRITE_DATA . . . 677
inbound_mailbox_7.READ_DATA . . . 677
inbound_mailbox_7.STATUS . . . 677
inbound_mailbox_7.ERROR_FLAGS . . . 677
inbound_mailbox_7.WIRQT . . . 677
inbound_mailbox_7.RIRQT . . . 678
inbound_mailbox_7.IRQS . . . 678
inbound_mailbox_7.IRQEN . . . 678
inbound_mailbox_7.IRQP . . . 678
inbound_mailbox_7.CTRL . . . 679
System Timer OCTS . . . 679
Overview . . . 679
Operation . . . 679
Counting and Synchronization . . . 680
Pulse Generation and Clock Crossing . . . 681
Status and Credit Expiration . . . 681
Reset . . . 681
Interface . . . 681
Ports . . . 681
OCTS Synchronization Interface . . . 682
Timer Count . . . 682
Timing . . . 682
Memory Map and Register Reference . . . 682
Memory Map . . . 682
Detailed Register Map . . . 682
OCTS System Timer Address Map . . . 682
TIMER_START . . . 683
CTRL . . . 683
STATUS . . . 683
TIMER_PRESET_LO . . . 683
TIMER_PRESET_HI . . . 684
TIMER_COUNT_LO . . . 684
TIMER_COUNT_HI . . . 684
CREDIT_EXPIRED . . . 684
TIMER_GPIO_ENABLE . . . 684
eFuse . . . 684
Overview . . . 685

Description and Features . . . 685
Architecture . . . 685
Interface Controller . . . 685
Access Paths . . . 685
Command-path ownership . . . 685
Destination Endpoints . . . 685
Shadow Register Subsystem . . . 686
Features . . . 686
State Machine Architecture . . . 686
Parameterized Command Typedefs . . . 686
Foundry-Specific Signals . . . 687
Interface SHIM Layer . . . 687
eFuse Bank Model . . . 688
Access Permissions and Security . . . 688
Hardware vs. Software Locks . . . 688
Security Disable Bypass Mechanism . . . 689
Lifecycle State (LC_STATE) Effects . . . 689
Secure Test Mode (secure_tm_i) . . . 690
Interface . . . 691
Configuration Parameters . . . 691
Main AXI4-Lite Interfaces . . . 691
Fuse Bank Control Interface . . . 691
Fuse Command Interface . . . 691
Security and Control Signals . . . 691
Fuse Command Protocol . . . 691
Memory Map and Register Reference . . . 692
Memory Map . . . 692
Detailed Register Map . . . 693
Shadow Registers . . . 693
Efuse address map of physical efuse macro . . . 693
LOCKS . . . 693
JTAG_PUBLIC_IDENTITY . . . 695
I2C_I3C_ID[9] . . . 695
SMC_CONFIG . . . 695
OCCP_TRANSPORT_TIMEOUT . . . 696
SPARE[28] . . . 696
MMR Registers . . . 696
eFuse Token Registers Address Map . . . 696
RMA_SIP_TOKEN_I[8] . . . 697
RMA_CHIPLET_TOKEN_I[8] . . . 697
SEC_DISABLE_TOKEN_I[8] . . . 697
TOKEN_EOP . . . 697
RMA_SIP_TOKEN_MATCH . . . 698
RMA_CHIPLET_TOKEN_MATCH . . . 698

SEC_DISABLE_TOKEN_MATCH . . . 698
TOKEN_MATCH_FAULT . . . 698
eFuse Control Registers . . . 698
eFuse Interface Control Address Map . . . 699
EFUSE_INTERFACE_CTRL_STATUS . . . 699
EFUSE_PROGRAM_CTRL . . . 699
EFUSE_READ_CTRL . . . 700
EFUSE_PROGRAM_INTERFACE_READ_DATA . . . 700
EFUSE_READ_INTERFACE_READ_DATA . . . 700
EFUSE_READ_REQ_TIMEOUT . . . 701
EFUSE_PROGRAM_REQ_TIMEOUT . . . 701
SHIM Control Registers . . . 701
eFuse SHIM Control Model Address Map . . . 701
EFUSE_BANK_INIT_TIME . . . 701
UART 16550 . . . 701
Overview . . . 701
Architecture . . . 702
Transmit and Receive Paths . . . 703
Baud Timing and Register Views . . . 703
FIFO Protection and Line Errors . . . 703
Interrupt Priority and Service . . . 703
Interface . . . 704
Configuration Parameters . . . 704
Modem Control Interface . . . 704
DMA Interface . . . 704
Interface Timing Considerations . . . 704
Clock Domain . . . 704
Serial Interface Timing . . . 705
Protocol Support . . . 705
Memory Map and Register Reference . . . 705
Memory Map . . . 705
Detailed Register Map . . . 705
UART 16550 Main Address Map . . . 705
RBR . . . 706
IER . . . 706
IIR . . . 706
LCR . . . 706
MCR . . . 707
LSR . . . 707
MSR . . . 708
SCR . . . 708
ECR . . . 708
ITR . . . 709
UART 16550 Main Write-Only Address Map . . . 709

THR . . . 709
FCR . . . 709
UART 16550 Divisor Latch Address Map . . . 710
DLL . . . 710
DLM . . . 710
Log Engine . . . 710
Overview . . . 710
Operation . . . 712
Memory Slots and Lengths . . . 712
Arbitration and Data Transfer . . . 712
Errors and Interrupts . . . 712
Reset and Disable . . . 713
Interface . . . 713
Memory Map and Register Reference . . . 713
Memory Map . . . 713
Detailed Register Map . . . 714
Log Engine Address Map . . . 714
CTRL . . . 714
LOG_REGION_SIZE . . . 714
LOG_REGION_ADDR . . . 715
LOG_WRITE_ADDR . . . 715
INTR_STATUS . . . 715
INTR_ENABLE . . . 715
INTR_TEST . . . 715
LOG_CTRL[16] . . . 716
I2C Controller . . . 716
OCAH Modifications . . . 717
Memory Map and Register Reference . . . 719
Memory Map . . . 719
Address Alignment and Access Width . . . 719
Detailed Register Map . . . 719
I2C Address Map . . . 719
INTR_STATE . . . 720
INTR_ENABLE . . . 721
INTR_TEST . . . 722
SMBUS_CTRL . . . 722
CTRL . . . 722
STATUS . . . 724
RDATA . . . 724
FDATA . . . 724
FIFO_CTRL . . . 725
HOST_FIFO_CONFIG . . . 725
TARGET_FIFO_CONFIG . . . 725
HOST_FIFO_STATUS . . . 725

TARGET_FIFO_STATUS . . . 726
OVRD . . . 726
VAL . . . 726
TIMING0 . . . 726
TIMING1 . . . 727
TIMING2 . . . 727
TIMING3 . . . 727
TIMING4 . . . 727
TIMEOUT_CTRL . . . 728
TARGET_ID . . . 728
ACQDATA . . . 728
TXDATA . . . 729
HOST_TIMEOUT_CTRL . . . 729
TARGET_TIMEOUT_CTRL . . . 729
TARGET_NACK_COUNT . . . 730
TARGET_ACK_CTRL . . . 730
ACQ_FIFO_NEXT_DATA . . . 730
HOST_NACK_HANDLER_TIMEOUT . . . 730
CONTROLLER_EVENTS . . . 730
TARGET_EVENTS . . . 731
SMBUS_STATUS . . . 731
I2C Control Address Map . . . 731
I2C_CTRL[3] . . . 732
I2C Wrapper Address Map . . . 732
i2c[3].INTR_STATE . . . 734
i2c[3].INTR_ENABLE . . . 735
i2c[3].INTR_TEST . . . 736
i2c[3].SMBUS_CTRL . . . 737
i2c[3].CTRL . . . 737
i2c[3].STATUS . . . 738
i2c[3].RDATA . . . 739
i2c[3].FDATA . . . 739
i2c[3].FIFO_CTRL . . . 739
i2c[3].HOST_FIFO_CONFIG . . . 740
i2c[3].TARGET_FIFO_CONFIG . . . 740
i2c[3].HOST_FIFO_STATUS . . . 740
i2c[3].TARGET_FIFO_STATUS . . . 740
i2c[3].OVRD . . . 740
i2c[3].VAL . . . 741
i2c[3].TIMING0 . . . 741
i2c[3].TIMING1 . . . 741
i2c[3].TIMING2 . . . 742
i2c[3].TIMING3 . . . 742
i2c[3].TIMING4 . . . 742

i2c[3].TIMEOUT_CTRL . . . 742
i2c[3].TARGET_ID . . . 743
i2c[3].ACQDATA . . . 743
i2c[3].TXDATA . . . 743
i2c[3].HOST_TIMEOUT_CTRL . . . 744
i2c[3].TARGET_TIMEOUT_CTRL . . . 744
i2c[3].TARGET_NACK_COUNT . . . 744
i2c[3].TARGET_ACK_CTRL . . . 744
i2c[3].ACQ_FIFO_NEXT_DATA . . . 745
i2c[3].HOST_NACK_HANDLER_TIMEOUT . . . 745
i2c[3].CONTROLLER_EVENTS . . . 745
i2c[3].TARGET_EVENTS . . . 745
i2c[3].SMBUS_STATUS . . . 746
I2C_CTRL[3] . . . 746
I3C Controller . . . 746
OCAH Modifications . . . 747
Memory Map and Register Reference . . . 749
Memory Map . . . 749
Integration Limits . . . 749
Detailed Register Map . . . 749
OCA I3C Controller Wrapper Address Map . . . 749
i3c_csr[6].I3CBase.HCI_VERSION . . . 761
i3c_csr[6].I3CBase.HC_CONTROL . . . 761
i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR . . . 762
i3c_csr[6].I3CBase.HC_CAPABILITIES . . . 762
i3c_csr[6].I3CBase.RESET_CONTROL . . . 764
i3c_csr[6].I3CBase.PRESENT_STATE . . . 764
i3c_csr[6].I3CBase.INTR_STATUS . . . 764
i3c_csr[6].I3CBase.INTR_STATUS_ENABLE . . . 765
i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE . . . 765
i3c_csr[6].I3CBase.INTR_FORCE . . . 765
i3c_csr[6].I3CBase.DAT_SECTION_OFFSET . . . 766
i3c_csr[6].I3CBase.DCT_SECTION_OFFSET . . . 766
i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET . . . 766
i3c_csr[6].I3CBase.PIO_SECTION_OFFSET . . . 766
i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET . . . 766
i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN . . . 767
i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL . . . 767
i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL . . . 767
i3c_csr[6].I3CBase.DEV_CTX_BASE_LO . . . 768
i3c_csr[6].I3CBase.DEV_CTX_BASE_HI . . . 768
i3c_csr[6].I3CBase.DEV_CTX_SG . . . 768
i3c_csr[6].PIOControl.COMMAND_PORT . . . 768
i3c_csr[6].PIOControl.RESPONSE_PORT . . . 768

i3c_csr[6].PIOControl.TX_DATA_PORT . . . 769
i3c_csr[6].PIOControl.RX_DATA_PORT . . . 769
i3c_csr[6].PIOControl.IBI_PORT . . . 769
i3c_csr[6].PIOControl.QUEUE_THLD_CTRL . . . 769
i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL . . . 769
i3c_csr[6].PIOControl.QUEUE_SIZE . . . 770
i3c_csr[6].PIOControl.ALT_QUEUE_SIZE . . . 770
i3c_csr[6].PIOControl.PIO_INTR_STATUS . . . 770
i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE . . . 770
i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE . . . 771
i3c_csr[6].PIOControl.PIO_INTR_FORCE . . . 771
i3c_csr[6].PIOControl.PIO_CONTROL . . . 771
i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER . . . 772
i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0 . . . 772
i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1 . . . 772
i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2 . . . 772
i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3 . . . 773
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0 . . . 773
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1 . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2 . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3 . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4 . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5 . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED . . . 774
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0 . . . 775
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1 . . . 776
i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET . . . 776
i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL . . . 777
i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS . . . 777
i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS . . . 777
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0 . . . 778
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1 . . . 778
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0 . . . 778
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1 . . . 779
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2 . . . 779
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3 . . . 779
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4 . . . 779
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED . . . 779
i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA . . . 780
i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER . . . 780
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL . . . 780
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR . . . 781
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES . . . 781
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR . . . 782

i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS . . . 783
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR . . . 783
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO . . . 784
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS . . . 784
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO . . . 784
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE . . . 785
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE . . . 785
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS . . . 785
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS . . . 786
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR . . . 786
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL . . . 786
i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL . . . 787
i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER . . . 787
i3c_csr[6].I3C_EC.TTI.CONTROL . . . 787
i3c_csr[6].I3C_EC.TTI.STATUS . . . 788
i3c_csr[6].I3C_EC.TTI.RESET_CONTROL . . . 788
i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS . . . 788
i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH . . . 789
i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH . . . 789
i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH . . . 789
i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS . . . 789
i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE . . . 791
i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE . . . 791
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL . . . 792
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS . . . 792
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE . . . 793
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE . . . 793
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5 . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING . . . 794
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC . . . 795
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH . . . 795
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY . . . 795
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED . . . 795
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW . . . 795
i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW . . . 795
i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT . . . 796
i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT . . . 796
i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT . . . 796
i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT . . . 796

i3c_csr[6].I3C_EC.TTI.IBI_PORT . . . 796
i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE . . . 796
i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE . . . 796
i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL . . . 797
i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL . . . 797
i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER . . . 797
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL . . . 797
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2 . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3 . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF . . . 798
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR . . . 799
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2 . . . 800
i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3 . . . 800
i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG . . . 800
i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG . . . 800
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG . . . 800
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG . . . 801
i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG . . . 802
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG . . . 803
i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG . . . 804
i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG . . . 804
i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG . . . 804
i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER . . . 804
i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG . . . 804

i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER . . . 804
i3c_csr[6].DAT.DAT_MEMORY[128] . . . 805
i3c_csr[6].DCT.DCT_MEMORY[128] . . . 805
AVSBus Controller . . . 805
Overview . . . 805
Description and Features . . . 806
Architecture . . . 806
AVS Protocol Engine . . . 807
Clock Generation and Management . . . 808
CRC Verification Engine . . . 808
Command and Response FIFO System . . . 808
Reset and Initialization . . . 809
Data Flow and Control Path . . . 809
Command Path . . . 809
Response Inspection . . . 809
Clock Domain Architecture . . . 809
Protocol State Machine . . . 809
Error Handling and Diagnostics . . . 811
Interface . . . 812
AVSBus Frame Format . . . 812
Clock Architecture . . . 813
Reset Architecture . . . 814
Interrupt Interface . . . 814
Pad Enable Interface . . . 815
Timing . . . 815
Integration Example . . . 815
Memory Map and Register Reference . . . 816
Memory Map . . . 816
Address Alignment and Access Requirements . . . 816
Detailed Register Map . . . 817
AVSBus Controller Address Map . . . 817
AVS_CMD . . . 817
AVS_READBACK . . . 818
AVS_DEBUG_READBACK . . . 819
AVS_LATEST_SLAVE_SUBFRAME . . . 819
AVS_NORMAL_STATUS . . . 819
AVS_SLAVE_STATUS . . . 819
AVS_FIFOS_STATUS . . . 820
AVS_INTERRUPT . . . 820
AVS_INTERRUPT_MASK . . . 820
AVS_INTERRUPT_CLEAR . . . 821
AVS_CFG_0 . . . 821
AVS_CFG_1 . . . 821
AVS_CONFIG . . . 822

GPIO Peripheral . . . 822
Overview . . . 822
Operation . . . 823
Data and Direction Ownership . . . 824
Input Readback . . . 825
SMC Pad Ring . . . 825
Pad Settings and Secondary Function . . . 826
Interrupts . . . 827
Reset and Strap Capture . . . 827
Register Access Filtering . . . 827
Interface . . . 827
LSIO Interface . . . 828
LSIO Protocol Specifications . . . 828
GPIO Pad Interface . . . 828
Interrupt Interface . . . 828
Interrupt Specifications . . . 828
Memory Map and Register Reference . . . 828
Memory Map . . . 828
Reset Values . . . 828
Detailed Register Map . . . 829
GPIO Interface Address Map . . . 829
DATA_CTRL . . . 829
DATA_CTRL_ENABLE . . . 830
ACCESS_FILTER . . . 830
GPIO Control Address Map . . . 830
CONTROL . . . 830
GPIO Wrapper Address Map . . . 831
DATA_CTRL . . . 831
DATA_CTRL_ENABLE . . . 832
ACCESS_FILTER . . . 832
CONTROL . . . 833
Telemetry Receiver . . . 833
Overview . . . 833
Message Format . . . 833
Architecture . . . 834
Assembly and Decode . . . 834
Message Queue and Readout . . . 834
Interrupts . . . 834
Reset and Flush . . . 835
Interface . . . 835
Configuration Parameters . . . 835
Signal Interfaces . . . 835
Global Interface . . . 835
Clock and Reset . . . 835

ATB Telemetry Interface . . . 835
ATB Protocol Specifications . . . 836
Interrupt Interface . . . 836
Interrupt Specifications . . . 836
Subsystem Wrapper . . . 836
Memory Map and Register Reference . . . 836
Memory Map . . . 836
Detailed Register Map . . . 836
Telemetry Receiver Address Map . . . 836
CTRL . . . 837
STATUS . . . 837
INTR_STATUS . . . 838
INTR_ENABLE . . . 838
INTR_TEST . . . 838
TELEMETRY_PROBE_ID . . . 838
TELEMETRY_COUNTER_VLDS . . . 838
TELEMETRY_COUNTER[32] . . . 839
Telemetry Receiver Wrapper Address Map . . . 839
telemetry_receiver[3].CTRL . . . 839
telemetry_receiver[3].STATUS . . . 840
telemetry_receiver[3].INTR_STATUS . . . 840
telemetry_receiver[3].INTR_ENABLE . . . 840
telemetry_receiver[3].INTR_TEST . . . 841
telemetry_receiver[3].TELEMETRY_PROBE_ID . . . 841
telemetry_receiver[3].TELEMETRY_COUNTER_VLDS . . . 841
telemetry_receiver[3].TELEMETRY_COUNTER[32] . . . 841
UART and Log Engine Wrapper Registers . . . 841
UART Wrapper Address Map . . . 841
uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL . . . 843
uart_log_engine_wrap[4].uart.RBR . . . 843
uart_log_engine_wrap[4].uart.IER . . . 843
uart_log_engine_wrap[4].uart.IIR . . . 844
uart_log_engine_wrap[4].uart.LCR . . . 844
uart_log_engine_wrap[4].uart.MCR . . . 844
uart_log_engine_wrap[4].uart.LSR . . . 845
uart_log_engine_wrap[4].uart.MSR . . . 845
uart_log_engine_wrap[4].uart.SCR . . . 846
uart_log_engine_wrap[4].uart.ECR . . . 846
uart_log_engine_wrap[4].uart.ITR . . . 846
uart_log_engine_wrap[4].log_engine.CTRL . . . 846
uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE . . . 847
uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR . . . 847
uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR . . . 847
uart_log_engine_wrap[4].log_engine.INTR_STATUS . . . 847

uart_log_engine_wrap[4].log_engine.INTR_ENABLE . . . 847
uart_log_engine_wrap[4].log_engine.INTR_TEST . . . 848
uart_log_engine_wrap[4].log_engine.LOG_CTRL[16] . . . 848
UART Log Engine Control Address Map . . . 848
CTRL . . . 848
UART Log Engine Wrapper Address Map . . . 848
uart_log_engine_ctrl.CTRL . . . 849
RBR . . . 849
IER . . . 849
IIR . . . 850
LCR . . . 850
MCR . . . 850
LSR . . . 851
MSR . . . 851
SCR . . . 852
ECR . . . 852
ITR . . . 852
log_engine.CTRL . . . 853
LOG_REGION_SIZE . . . 853
LOG_REGION_ADDR . . . 853
LOG_WRITE_ADDR . . . 853
INTR_STATUS . . . 853
INTR_ENABLE . . . 854
INTR_TEST . . . 854
LOG_CTRL[16] . . . 854
Peripheral Parameter Settings . . . 854
2.1.6.8. Design-for-Debug (DFD) and Core Logic Analyzer (CLA) . . . 855
Debug Bus . . . 856
Core Logic Analyzer . . . 856
Actions and Cross-Triggering . . . 857
Trace . . . 857
Interrupts . . . 857
Memory Map and Register Reference . . . 858
Memory Map . . . 858
Detailed Register Map . . . 858
SMC DFD MMR Aperture Address Map . . . 858
Trdstramcontrol . . . 863
Trdstramimpl . . . 864
Trdstramstartlow . . . 864
Trdstramstarthigh . . . 864
Trdstramlimitlow . . . 864
Trdstramlimithigh . . . 864
Trdstramwplow . . . 865
Trdstramwphigh . . . 865

Trdstramrplow . . . 865
Trdstramrphigh . . . 865
Trdstramdata . . . 865
dst_sink.ScratchLo . . . 865
dst_sink.ScratchHi . . . 866
Trfunnelcontrol . . . 866
Trfunnelimpl . . . 866
Trfunneldisinput . . . 866
funnel.ScratchLo . . . 866
funnel.ScratchHi . . . 866
CDbgClaCounter0Cfg . . . 867
CDbgClaCounter1Cfg . . . 867
CDbgClaCounter2Cfg . . . 867
CDbgClaCounter3Cfg . . . 867
CDbgNode0Eap0 . . . 868
CDbgNode0Eap1 . . . 868
CDbgNode1Eap0 . . . 869
CDbgNode1Eap1 . . . 869
CDbgNode2Eap0 . . . 870
CDbgNode2Eap1 . . . 870
CDbgNode3Eap0 . . . 871
CDbgNode3Eap1 . . . 871
CDbgSignalMask0Lo . . . 872
CDbgSignalMatch0Lo . . . 872
CDbgSignalMask1Lo . . . 872
CDbgSignalMatch1Lo . . . 872
CDbgSignalEdgeDetectCfg . . . 872
CDbgEapStatus . . . 873
CDbgClaCtrlStatus . . . 874
CDbgMuxSelLo . . . 874
CDbgRsvd1 . . . 875
CDbgRsvd2 . . . 875
CDbgTransitionMaskLo . . . 875
CDbgTransitionFromValueLo . . . 875
CDbgTransitionToValueLo . . . 875
CDbgOnesCountMaskLo . . . 876
CDbgOnesCountValue . . . 876
CDbgAnyChangeLo . . . 876
CDbgSignalSnapshotNode0Eap0Lo . . . 876
CDbgSignalSnapshotNode0Eap1Lo . . . 876
CDbgSignalSnapshotNode1Eap0Lo . . . 876
CDbgSignalSnapshotNode1Eap1Lo . . . 876
CDbgSignalSnapshotNode2Eap0Lo . . . 876
CDbgSignalSnapshotNode2Eap1Lo . . . 877

CDbgSignalSnapshotNode3Eap0Lo . . . 877
CDbgSignalSnapshotNode3Eap1Lo . . . 877
CDbgClaTimeMatch . . . 877
CDbgSignalMask2Lo . . . 877
CDbgSignalMatch2Lo . . . 877
CDbgSignalMask3Lo . . . 877
CDbgSignalMatch3Lo . . . 878
CDbgNode0Eap2 . . . 878
CDbgNode0Eap3 . . . 878
CDbgNode1Eap2 . . . 879
CDbgNode1Eap3 . . . 879
CDbgNode2Eap2 . . . 880
CDbgNode2Eap3 . . . 880
CDbgNode3Eap2 . . . 881
CDbgNode3Eap3 . . . 881
CDbgSignalSnapshotNode0Eap2Lo . . . 882
CDbgSignalSnapshotNode0Eap3Lo . . . 882
CDbgSignalSnapshotNode1Eap2Lo . . . 882
CDbgSignalSnapshotNode1Eap3Lo . . . 882
CDbgSignalSnapshotNode2Eap2Lo . . . 882
CDbgSignalSnapshotNode2Eap3Lo . . . 882
CDbgSignalSnapshotNode3Eap2Lo . . . 883
CDbgSignalSnapshotNode3Eap3Lo . . . 883
CDbgSignalDelayMuxSel . . . 883
CDbgClaXtriggerTimestretch . . . 883
CDbgClaTimestamp . . . 883
CDbgClaTimestampSync . . . 884
CDbgClaTimestampConfig . . . 884
CDbgClaTimestampOffset . . . 884
CDbgSignalMask0Hi . . . 884
CDbgSignalMatch0Hi . . . 884
CDbgSignalMask1Hi . . . 884
CDbgSignalMatch1Hi . . . 885
CDbgSignalMask2Hi . . . 885
CDbgSignalMatch2Hi . . . 885
CDbgSignalMask3Hi . . . 885
CDbgSignalMatch3Hi . . . 885
CDbgMuxSelHi . . . 885
CDbgTransitionMaskHi . . . 886
CDbgTransitionFromValueHi . . . 886
CDbgTransitionToValueHi . . . 886
CDbgOnesCountMaskHi . . . 886
CDbgAnyChangeHi . . . 886
CDbgSignalSnapshotNode0Eap0Hi . . . 886

CDbgSignalSnapshotNode0Eap1Hi . . . 886
CDbgSignalSnapshotNode1Eap0Hi . . . 887
CDbgSignalSnapshotNode1Eap1Hi . . . 887
CDbgSignalSnapshotNode2Eap0Hi . . . 887
CDbgSignalSnapshotNode2Eap1Hi . . . 887
CDbgSignalSnapshotNode3Eap0Hi . . . 887
CDbgSignalSnapshotNode3Eap1Hi . . . 887
CDbgSignalSnapshotNode0Eap2Hi . . . 887
CDbgSignalSnapshotNode0Eap3Hi . . . 888
CDbgSignalSnapshotNode1Eap2Hi . . . 888
CDbgSignalSnapshotNode1Eap3Hi . . . 888
CDbgSignalSnapshotNode2Eap2Hi . . . 888
CDbgSignalSnapshotNode2Eap3Hi . . . 888
CDbgSignalSnapshotNode3Eap2Hi . . . 888
CDbgSignalSnapshotNode3Eap3Hi . . . 888
CDbgLfsr . . . 888
CDbgLfsrMask . . . 889
CDbgTimestampCapture . . . 889
CDbgCompare0Lo . . . 889
CDbgCompare0MaskLo . . . 889
CDbgCompare1Lo . . . 889
CDbgCompare1MaskLo . . . 889
CDbgCompare2Lo . . . 889
CDbgCompare2MaskLo . . . 890
CDbgCompare3Lo . . . 890
CDbgCompare3MaskLo . . . 890
CDbgCompare0Hi . . . 890
CDbgCompare0MaskHi . . . 890
CDbgCompare1Hi . . . 890
CDbgCompare1MaskHi . . . 890
CDbgCompare2Hi . . . 891
CDbgCompare2MaskHi . . . 891
CDbgCompare3Hi . . . 891
CDbgCompare3MaskHi . . . 891
Scratch . . . 891
Trdstcontrol . . . 891
Trdstimpl . . . 892
Trdstinstfeatures . . . 892
CDbgDebugTraceCfg . . . 893
dst.ScratchLo . . . 893
dst.ScratchHi . . . 893
2.1.6.9. eFuse Shadow Register Scan Protection . . . 893
SMC eFuse Asset Classification . . . 893
Implementation . . . 894

Downstream Lock Path . . . 895
Notes for DFT . . . 895
Verification . . . 895
2.1.6.10. Memory Map and Register Reference . . . 895
Memory Map . . . 895
Address Space Organization . . . 895
AXI-Lite External Window . . . 901
Register Interface Standards . . . 902
Access Control and Security . . . 902
Detailed Register Map . . . 902
External Window Reference Maps . . . 902
PLL Wrapper Placeholder Address Map . . . 902
mem_array[954] . . . 902
PVT Wrapper Placeholder Address Map . . . 903
mem_array[594] . . . 903
Straps Address Map . . . 903
STRAPS_LO . . . 903
STRAPS_HI . . . 904
Spare SMC Register Blocks . . . 904
Chip Config, Scratch, NDM Reset, and Misc Wrap . . . 904
Chip Configuration Address Map . . . 904
VERSION_LO . . . 904
VERSION_HI . . . 904
CHIP_ID . . . 904
LC_STATE . . . 905
Scratch Address Map . . . 905
SCRATCH[8] . . . 905
NDM Reset Control Address Map . . . 905
NDMRESET_REQUEST . . . 905
NDMRESET_PROCESS . . . 905
NDMRESET_CLUSTER_COUNT . . . 906
SMC Miscellaneous Wrapper Address Map . . . 906
scratch_cold.SCRATCH[8] . . . 906
scratch_cold_warm.SCRATCH[8] . . . 906
VERSION_LO . . . 907
VERSION_HI . . . 907
CHIP_ID . . . 907
LC_STATE . . . 907
NDMRESET_REQUEST . . . 907
NDMRESET_PROCESS . . . 907
NDMRESET_CLUSTER_COUNT . . . 907
Base Config and DFX Status . . . 908
SMC Base Configuration Address Map . . . 908
GLOBAL_BASE . . . 908

LOCAL_BASE . . . 908
REGION_SIZE . . . 908
CLOCK_GATE_CONTROL . . . 909
HANG_DET_SYS_AXI_CTRL . . . 909
HANG_DET_SYS_AXI_TIMEOUT_THRESHOLD . . . 910
HANG_DET_SEP_AXI_CTRL . . . 910
HANG_DET_SEP_AXI_TIMEOUT_THRESHOLD . . . 910
HANG_DET_DATA_ACCEL_CTRL . . . 911
HANG_DET_DATA_ACCEL_TIMEOUT_THRESHOLD . . . 911
DFX Control and Status Address Map . . . 911
STATUS_SMU . . . 912
DEBUG_CTRL . . . 912
DEBUG_BUS_MUX . . . 912
Component Register References . . . 913
2.1.7. Debug & Test Ports (DTP) . . . 914
2.1.7.1. Capabilities . . . 914
2.1.7.2. Overview . . . 914
JTAG Interface Unit . . . 916
Cross Trigger Network . . . 916
Debug Access Paths . . . 916
2.1.7.3. Clocks and Resets . . . 917
Clock Domains . . . 917
Reset Architecture . . . 917
IC_RESET Overrides . . . 917
Clock Stop . . . 918
Halt Path Architecture . . . 918
Signal Flow . . . 920
Clock Distribution . . . 920
Reset Distribution . . . 921
2.1.7.4. JTAG Operation . . . 923
DTP JTAG Topology . . . 923
STAP Secondary Scan Path . . . 924
JTAG2AXI Bridges . . . 924
Debug Disable . . . 924
2.1.7.5. Cross-Trigger Operation . . . 925
Event Routing . . . 925
Clock Stop . . . 925
Programming Reference . . . 925
2.1.7.6. IP Reference . . . 926
JTAG . . . 926
JTAG Interface Unit . . . 926
Overview . . . 926
Configuration Parameters . . . 926
Architecture . . . 926

Scan Chain Topology . . . 927
STAP Chain Hierarchy . . . 927
Debug Disable Synchronization . . . 928
iJTAG Network . . . 928
DFT Secure Chain . . . 929
DFT Non-Secure Chain . . . 929
DFD Chain . . . 929
JTAG2AXI Interfaces . . . 929
Interface . . . 929
Instruction Encodings . . . 929
Signal Interface . . . 931
Dependencies . . . 931
JTAG PTAP . . . 931
Overview . . . 931
Configuration Parameters . . . 932
Architecture . . . 932
Module Hierarchy . . . 934
TAP Controller State Machine . . . 934
Reset Hierarchy . . . 935
TDR Multiplexer . . . 936
Zero-Length Bypass . . . 936
TDO Retiming . . . 936
ID Code . . . 936
Debug Control . . . 937
JTAG Capabilities . . . 937
Boundary Scan Support . . . 938
BIST Support . . . 938
Test Mode Persistence (TMP) Support . . . 938
3DIC Support . . . 939
IC_RESET Support . . . 939
iJTAG Support . . . 940
JTAG2AXI Support . . . 940
*_JTAG2AXI_CAPS . . . 940
*_AXI_SINGLE_OP . . . 941
*_AXI_SERIES_CTRL . . . 942
*_AXI_SERIES_DATA_INCR . . . 943
*_AXI_SERIES_DATA_NO_INCR . . . 944
*_AXI_SERIES_DATA_WITH_ERROR_STATUS . . . 946
System Reset Recovery . . . 947
Debug Disable . . . 947
Interface . . . 947
Instruction Encodings . . . 947
Signal Interface . . . 949
Dependencies . . . 949

JTAG STAP . . . 949
Overview . . . 949
Configuration Parameters . . . 950
Architecture . . . 950
Module Hierarchy . . . 950
STAP Module . . . 950
3DCR Register . . . 951
STAP Types . . . 952
Debug Disable . . . 952
Lockup Latches . . . 952
TDO Lockup Latch . . . 953
TDI Lockup Latch . . . 953
Scan Output Lockup Latch . . . 953
Interface . . . 953
Signal Interface . . . 953
Timing Requirements . . . 953
Dependencies . . . 953
Cross-Trigger . . . 953
Cross-Trigger Network . . . 953
Overview . . . 954
Configuration Parameters . . . 954
Architecture . . . 954
Cross Trigger Port (CTP) . . . 955
Wire-OR Mode . . . 955
Point-to-Point Mode . . . 956
Cross Trigger Matrix (CTM) . . . 956
Address Map . . . 957
Internal vs External CTP Architecture . . . 957
Internal CTP Signal Remapping . . . 957
Signal Flow . . . 958
Clock Stop Control . . . 958
Port Indexing . . . 959
Interface . . . 959
Cross Trigger Port (CTP) . . . 959
Cross Trigger Matrix (CTM) . . . 959
Cross Trigger Network Module Interface . . . 959
Dependencies . . . 960
Programming . . . 960
Memory Map . . . 960
Memory Map . . . 960
Detailed Register Map . . . 960
Cross Trigger Matrix Registers . . . 960
Cross Trigger Matrix Address Map . . . 960
CT_SRC[26].CONFIG_0 . . . 961

Cross Trigger Port Registers . . . 961
Cross Trigger Port Address Map . . . 961
CONFIG . . . 961
STATUS . . . 962
STRETCH_MULT . . . 962
Composite Network Registers . . . 962
Cross Trigger Network Address Map . . . 962
CT_SRC[26].CONFIG_0 . . . 963
ctp[16].CONFIG . . . 963
ctp[16].STATUS . . . 963
ctp[16].STRETCH_MULT . . . 964
Cross-Trigger Port . . . 964
Overview . . . 964
Configuration . . . 965
Architecture . . . 965
Submodule Hierarchy . . . 966
Wire-OR Mode . . . 967
Pulse Stretcher Operation . . . 967
Point-to-Point Mode . . . 968
Synchronization and CDC Handling . . . 970
Handshake State Machines . . . 971
Interface . . . 972
Signal Interface . . . 972
Dependencies . . . 972
Cross-Trigger Matrix . . . 972
Overview . . . 972
Architecture . . . 973
Routing Logic . . . 974
Port Indexing . . . 974
Interface . . . 974
Signal Interface . . . 974
Data Ports . . . 975
Dependencies . . . 975
2.1.7.7. Configuration and Reference . . . 975
Port Reference . . . 975
Configuration Parameters . . . 979
Address Map . . . 979
2.2. System Management Network (SMN) . . . 979
2.2.1. SMU Interface . . . 979
2.2.2. Addressing and Integration . . . 979
2.3. AXI-over-UCIe Bridge (AoU) . . . 980
2.3.1. Overview and Features . . . 980
2.3.1.1. Introduction . . . 980
Overview . . . 980

2.3.1.2. Features . . . 981
Basic features . . . 981
2.3.2. Architecture . . . 982
2.3.2.1. Block diagram . . . 982
2.3.2.2. TX_CORE . . . 983
TX_CORE QoS Scheme . . . 984
2.3.2.3. RX_CORE . . . 987
2.3.2.4. Message FIFO . . . 989
2.3.2.5. Data width converter . . . 990
Overview . . . 990
Width Adaptation Paths . . . 991
WLAST Regeneration . . . 992
2.3.2.6. AXI Transaction Splitter . . . 992
Overview . . . 992
Burst Splitting . . . 992
Response Reassembly . . . 993
SFR Configuration . . . 993
2.3.2.7. Write Early Response Controller . . . 994
Overview . . . 994
Transaction Flow -- Normal Case . . . 994
Transaction Flow -- Error Case . . . 995
ID Ordering . . . 995
Write-Outstanding Queue Backpressure . . . 996
Mixed Ordering -- Bufferable Before Non-Bufferable (Same ID) . . . 997
Mixed Ordering -- Non-Bufferable Before Bufferable (Same ID) . . . 999
Enable/Disable Procedure . . . 1000
SFR Fields . . . 1000
2.3.2.8. AXI Aggregator . . . 1001
Overview . . . 1001
Write Aggregation . . . 1001
Read Aggregation . . . 1002
SFR Fields . . . 1002
2.3.2.9. Credit Controller . . . 1003
2.3.2.10. RP Remapping . . . 1007
Configuration Rules . . . 1007
Header Encoding . . . 1008
Misconfiguration Behavior . . . 1008
2.3.2.11. UCIe Flit cancel mechanism . . . 1008
Cancel Cases . . . 1009
RX_FDI_IF Buffering . . . 1010
Timing . . . 1010
2.3.3. Interrupts and Errors . . . 1010
2.3.3.1. Interrupt . . . 1010
2.3.3.2. Error . . . 1012

2.3.4. PPA and Appendices . . . 1012
2.3.4.1. PPA . . . 1012
Performance . . . 1012
Power . . . 1015
Area . . . 1015
2.3.4.2. Appendix B. Referenced Documents . . . 1016
3. IP Reference . . . 1017
3.1. Management and I/O . . . 1017
3.2. Security . . . 1017
3.3. Debug and cross-trigger . . . 1017
3.4. Interconnect infrastructure . . . 1017
3.4.1. AXI Traffic Filter . . . 1017
3.4.1.1. Instances . . . 1018
3.4.1.2. Match, Then Permit . . . 1018
3.4.1.3. Address Range Granule . . . 1019
3.4.1.4. Blocked Transactions . . . 1019
3.4.1.5. Locking a Filter Entry . . . 1019
Appendix A: Register Field Access Notation . . . 1020
A.1. References . . . 1020
Appendix B: Referenced Documents . . . 1022
B.1. OCAH Documentation . . . 1022
B.2. OCA Specifications . . . 1022
B.3. Interface and Security Standards . . . 1022
B.4. External Design Documentation . . . 1023

# Documentation Overview

The Open Chiplet Atlas Harness (OCAH) is a chiplet design IP where the non-application-specific functions are integrated to form a chiplet design framework. These non-application-specific functions include common chiplet subsystems within the System Management Unit (SMU): Security Processor (SEP), System Management Controller (SMC), and Debug & Test Ports (DTP). The System Management Network (SMN) and AXI-over-UCIe bridge (AoU) are separate platform components.

This manual is the architecture and register-level reference for those blocks. Motivation, use cases, and the OCA-layer feature overview are on the OCAH documentation home page.

# List of Figures

Figure 1. OCAH Top-Level Block Diagram

Figure 2. OCAH Clock-Domain Zones

Figure 3. SMU composition with the optional SEP subsystem

Figure 4. SMU AXI crossbar connectivity

Figure 5. SMU aperture decode and routing

Figure 6. SMU clock distribution

Figure 7. SMU reset distribution

Figure 8. SEP lifecycle and feature-control propagation to SMC and DTP

Figure 9. SEP top-level microarchitecture

Figure 10. SEP clock tree and clock domains

Figure 11. SEP reset domains and reset sequencing

Figure 12. SEP Reset Controller reset logic

Figure 13. SEP crypto AXI isolation

Figure 14. SEP VeeR EL2 CPU subsystem and memory interfaces

Figure 15. SEP AXI4 connectivity. Filled circles indicate connections; dashes indicate no connection.

Figure 16. SEP input fabric

Figure 17. SEP local fabric

Figure 18. SEP output fabric

Figure 19. SEP system-peripheral traffic management

Figure 20. OpenTitan DMA core and inline hashing reference architecture

Figure 21. SEP secure DMA integration

Figure 22. SEP host mailbox architecture and interrupt routing

Figure 23. SEP peripheral extension port

Figure 24. SEP OTP fuse bank

Figure 25. eFuse controller and technology-specific interface shim

Figure 26. eFuse sensing state machine and shadow-register timing

Figure 27. SEP token processing

Figure 28. SEP lifecycle controller

Figure 29. SEP lifecycle state machine

Figure 30. SEP security domains

Figure 31. SEP cryptographic subsystem and external memory interfaces

Figure 32. SEP cryptographic subsystem resets

Figure 33. SEP TRNG and DRBG entropy distribution

Figure 34. Entropy source microarchitecture

Figure 35. Entropy source ring-oscillator complex

Figure 36. Entropy source decorrelator

Figure 37. OpenTitan AES core reference architecture

Figure 38. OpenTitan HMAC core reference architecture

Figure 39. OpenTitan KMAC core reference architecture

Figure 40. Adams Bridge ML-DSA reference architecture

Figure 41. Adams Bridge ML-KEM reference architecture

Figure 42. OpenTitan OTBN core reference architecture

Figure 43. Key Manager architecture

Figure 44. Key Manager cold and warm reset conditioning

Figure 45. Key Manager mailbox transport

Figure 46. Key Manager accelerator key path

Figure 47. SEP host and Key Manager key-load and key-transfer sequence

Figure 48. SRAM scrambler datapath

Figure 49. SMC top-level microarchitecture

Figure 50. SMC clock tree and clock domains in u_smc_base

Figure 51. SMC clock tree and clock domains in u_smc_cpu_wrapper and u_smc_peripherals

Figure 52. SMC reset unit architecture

Figure 53. SMC FLR isolation connections between chiplets

Figure 54. SMC-driven FLR isolation

Figure 55. Tile-request-driven FLR isolation

Figure 56. GPIO-driven FLR isolation

Figure 57. SMC CPU cluster and TileLink-to-AXI integration

Figure 58. SMC input fabric topology and routing

Figure 59. SMC local fabric and peripheral islands

Figure 60. SMC output fabric and remapping paths

Figure 61. SMC inbound and outbound traffic-filter mechanism

Figure 62. SMC DMA frontend, backend and shared fabric path

Figure 63. SMC memory zeroer state machine and AXI write flow

Figure 64. SMC mailbox wrapper and interrupt routing

Figure 65. System Timer OCTS primary/secondary synchronization and credit CDC

Figure 66. SMC UART wrapper

Figure 67. SMC log-engine wrapper and data paths

Figure 68. OpenTitan I2C core reference architecture

Figure 69. OCAH I2C wrapper and generated register interface

Figure 70. CHIPS Alliance i3c-core reference architecture

Figure 71. OCAH I3C wrapper
Figure 72. AVSBus controller block diagram
Figure 73. AVSBus protocol state machine
Figure 74. AVSBus clock selection
Figure 75. GPIO controller, interface shim, LSIO mux and strap-capture path
Figure 76. SMC pad ring GPIO interfaces and adopter pad shims
Figure 77. SMC DFD block diagram
Figure 78. DTP internal structure
Figure 79. DTP JTAG and CLA clock-stop aggregation
Figure 80. DTP system-clock and JTAG-clock distribution
Figure 81. DTP system and JTAG reset distribution
Figure 82. DTP JTAG scan chain
Figure 83. JTAG Interface Unit scan chain
Figure 84. PTAP module architecture
Figure 85. JTAG2AXI single-operation layout
Figure 86. JTAG2AXI series-control layout
Figure 87. JTAG2AXI incrementing data layout for effective size 0
Figure 88. JTAG2AXI incrementing data layout for effective size 1
Figure 89. JTAG2AXI incrementing data layout for effective size 2
Figure 90. JTAG2AXI incrementing data layout for effective size 3
Figure 91. JTAG2AXI non-incrementing data layout for effective size 0
Figure 92. JTAG2AXI non-incrementing data layout for effective size 1
Figure 93. JTAG2AXI non-incrementing data layout for effective size 2
Figure 94. JTAG2AXI non-incrementing data layout for effective size 3
Figure 95. JTAG2AXI error-status data layout for effective size 0
Figure 96. JTAG2AXI error-status data layout for effective size 1
Figure 97. JTAG2AXI error-status data layout for effective size 2
Figure 98. JTAG2AXI error-status data layout for effective size 3
Figure 99. STAP scan and control paths
Figure 100. SiP cross-trigger topology
Figure 101. CTP module structure
Figure 102. CTP wire-OR behavior
Figure 103. CTP wire-OR chiplet connections
Figure 104. CTP point-to-point behavior
Figure 105. CTP point-to-point chiplet connections
Figure 106. CTP request-acknowledge timing

Figure 107. CTM routing structure

Figure 108. Block Diagram of AOU_CORE and function description

Figure 109. Architecture of AOU_CORE

Figure 110. Block Diagram of Single RP AOU_TX_CORE

Figure 111. TX_CORE QoS Scheme

Figure 112. Round-Robin arbitration example

Figure 113. Port QoS arbitration example

Figure 114. AXI QoS arbitration example

Figure 115. Block Diagram of AOU_RX_CORE

Figure 116. Architecture of Chunk Decoder

Figure 117. Data FIFO

Figure 118. Req / Resp FIFO

Figure 119. AXI4MUX_3X1_TOP

Figure 120. AoU successful early write response

Figure 121. AoU error after an early write response

Figure 122. AoU write-outstanding backpressure

Figure 123. AoU bufferable then non-bufferable writes

Figure 124. AoU non-bufferable then bufferable writes

Figure 125. Write Aggregation -- Narrow-to-Wide Beat Merging

Figure 126. Block Diagram of AOU_CRD_CTRL

Figure 127. Example of WDATA and WSTRB FIFO Operation

Figure 128. Example of RDATA FIFO Operation

Figure 129. Resource Plane Remapping

Figure 130. UCIe Flit Cancel - Retransmit case

Figure 131. UCIe Flit Cancel - Flit Partially Valid case

Figure 132. UCIe Flit Cancel - Full Flit Cancel case

Figure 133. Example of selecting valid chunks for RX_CORE

Figure 134. Initial Latency of AOU_CORE with UCIe (Cycle)

Figure 135. AoU Bandwidth Efficiency with Respect to UCIe FDI Latency

Figure 136. Area of 1 RP AOU

# List of Tables

Table 1. OCAH address-map configuration layers

Table 2. Decode response codes

Table 3. OCAH Clock Domains

Table 4. RTL assertion macro families

Table 5. SMU AXI crossbar ports

Table 6. SMU AXI crossbar routing by initiator and decoded aperture

Table 7. SEP lifecycle and feature-control effects

Table 8. SEP clock inputs

Table 9. SEP reset boundaries

Table 10. SEP SW_RESET_N fields

Table 11. Paths isolated before each software reset

Table 12. SEP Reset Control Address Map register list

Table 13. sep_reset_ctrl.SW_RESET_N fields

Table 14. AxPROT driven by the SEP bus masters

Table 15. SEP processor configuration

Table 16. SEP closely coupled memories

Table 17. Scratch-bank reset behavior

Table 18. SEP Scratch Address Map register list

Table 19. sep_scratch.SCRATCH[8] fields

Table 20. SEP CPU Control Address Map register list

Table 21. sep_cpu_ctrl.CLOCK_GATE_CTRL fields

Table 22. sep_cpu_ctrl.REFERENCE_COUNTER fields

Table 23. sep_cpu_ctrl.TIMEOUT_INTERRUPT fields

Table 24. sep_cpu_ctrl.PKA_CTRL fields

Table 25. sep_cpu_ctrl.TIMEOUT_COUNT_DMA fields

Table 26. sep_cpu_ctrl.TIMEOUT_COUNT_SYS_IN fields

Table 27. sep_cpu_ctrl.TIMEOUT_COUNT_MAILBOX_INBOUND fields

Table 28. sep_cpu_ctrl.TIMEOUT_COUNT_MAILBOX_OUTBOUND fields

Table 29. sep_cpu_ctrl.TIMEOUT_COUNT_ENTROPY_WRITE fields

Table 30. sep_cpu_ctrl.TIMEOUT_COUNT_ENTROPY_READ fields

Table 31. sep_cpu_ctrl.TIMEOUT_COUNT_FILTER_OUT fields

Table 32. sep_cpu_ctrl.TIMEOUT_COUNT_ALIAS_REMAP fields

Table 33. sep_cpu_ctrl.TIMEOUT_ENABLE fields

Table 34. sep_cpu_ctrl.TIMEOUT_CLEAR fields

Table 35. sep_cpu_ctrl.TIMEOUT_MODE fields
Table 36. sep_cpu_ctrl.SEP_TEST_CTRL fields
Table 37. sep_cpu_ctrl.SEP_GLOBAL_BASE_ADDR fields
Table 38. sep_cpu_ctrl.SEP_LOCAL_BASE_ADDR fields
Table 39. sep_cpu_ctrl.SEP_REGION_SIZE fields
Table 40. sep_cpu_ctrl.SMU_GLOBAL_BASE_ADDR fields
Table 41. sep_cpu_ctrl.SMU_REGION_SIZE fields
Table 42. sep_cpu_ctrl.SMC_FUSE_SENSE_STATUS fields
Table 43. sep_cpu_ctrl.SEP_FUSE_SENSE_STATUS fields
Table 44. sep_cpu_ctrl.SEP_SW_DEBUG fields
Table 45. sep_cpu_ctrl.SEP_NMI_VEC fields
Table 46. sep_cpu_ctrl.SEP_NMI_VEC_LOCK fields
Table 47. sep_cpu_ctrl.EXT_TRNG_SRC_SEL fields
Table 48. sep_cpu_ctrl.EXT_TRNG_SRC_SEL_LOCK fields
Table 49. sep_cpu_ctrl.KM_WIPE_CTRL fields
Table 50. sep_cpu_ctrl.DMA_BUS_ERR_STATUS fields
Table 51. sep_cpu_ctrl.DMA_BUS_ERR_CLEAR fields
Table 52. sep_cpu_ctrl.PERIPH_BUS_ERR_STATUS fields
Table 53. sep_cpu_ctrl.PERIPH_BUS_ERR_CLEAR fields
Table 54. sep_cpu_ctrl.SEP_VERSION_ID fields
Table 55. SEP AXI4 connectivity terms
Table 56. SEP PIC source assignments
Table 57. SEP correctable-error counters
Table 58. SEP uncorrectable-error traps
Table 59. EL2_PIC register list
Table 60. el2_pic.meipl[256] fields
Table 61. el2_pic.meip[8] fields
Table 62. el2_pic.meie[256] fields
Table 63. el2_pic.mpiccfg fields
Table 64. el2_pic.meigwctrl[256] fields
Table 65. el2_pic.meigwclr[256] fields
Table 66. DMA Controller register list
Table 67. secure_dma.INTR_STATE fields
Table 68. secure_dma.INTR_ENABLE fields
Table 69. secure_dma.INTR_TEST fields
Table 70. secure_dma.ALERT_TEST fields

Table 71. secure_dma.SRC_ADDR_LO fields

Table 72. secure_dma.SRC_ADDR_HI fields

Table 73. secure_dma.DST_ADDR_LO fields

Table 74. secure_dma.DST_ADDR_HI fields

Table 75. secure_dma.ADDR_SPACE_ID fields

Table 76. secure_dma.ENABLED_MEMORY_RANGE_BASE fields

Table 77. secure_dma.ENABLED_MEMORY_RANGE_LIMIT fields

Table 78. secure_dma.RANGE_VALID fields

Table 79. secure_dma.RANGE_REGWEN fields

Table 80. secure_dma.CFG_REGWEN fields

Table 81. secure_dma.TOTAL_DATA_SIZE fields

Table 82. secure_dma.CHUNK_DATA_SIZE fields

Table 83. secure_dma.TRANSFER_WIDTH fields

Table 84. secure_dma.CONTROL fields

Table 85. secure_dma.SRC_CONFIG fields

Table 86. secure_dma.DST_CONFIG fields

Table 87. secure_dma.STATUS fields

Table 88. secure_dma.ERROR_CODE fields

Table 89. secure_dma.SHA2_DIGEST_0[16] fields

Table 90. secure_dma.HANDSHAKE_INTR_ENABLE fields

Table 91. secure_dma.CLEAR_INTR_SRC fields

Table 92. secure_dma.CLEAR_INTR_BUS fields

Table 93. secure_dma.INTR_SRC_ADDR_0[11] fields

Table 94. secure_dma.INTR_SRC_WR_VAL_0[11] fields

Table 95. SEP mailbox qualification

Table 96. Features

Table 97. Interrupts

Table 98. SPI Host register list

Table 99. spi_controller.INTR_STATE fields

Table 100. spi_controller.INTR_ENABLE fields

Table 101. spi_controller.INTR_TEST fields

Table 102. spi_controller.ALERT_TEST fields

Table 103. spi_controller.CONTROL fields

Table 104. spi_controller.STATUS fields

Table 105. spi_controller.CONFIGOPTS fields

Table 106. spi_controller.CSID fields

Table 107. spi_controller.COMMAND fields
Table 108. spi_controller.ERROR_ENABLE fields
Table 109. spi_controller.ERROR_STATUS fields
Table 110. spi_controller.EVENT_ENABLE fields
Table 111. Always-On Timer register list
Table 112. aon_timer.ALERT_TEST fields
Table 113. aon_timer.WKUP_CTRL fields
Table 114. aon_timer.WKUP_THOLD_HI fields
Table 115. aon_timer.WKUP_THOLD_LO fields
Table 116. aon_timer.WKUP_COUNT_HI fields
Table 117. aon_timer.WKUP_COUNT_LO fields
Table 118. aon_timer.WDOG_REGWEN fields
Table 119. aon_timer.WDOG_CTRL fields
Table 120. aon_timer.WDOG_BARK_THOLD fields
Table 121. aon_timer.WDOG_BITE_THOLD fields
Table 122. aon_timer.WDOG_COUNT fields
Table 123. aon_timer.INTR_STATE fields
Table 124. aon_timer.INTR_TEST fields
Table 125. aon_timer.WKUP_CAUSE fields
Table 126. SEP fuse fields
Table 127. SEP eFuse Map Address Map register list
Table 128. sep_efuse_map.LOCKS fields
Table 129. sep_efuse_map.LOCKS_SPARE fields
Table 130. sep_efuse_map.LC_STATE fields
Table 131. sep_efuse_map.SBOOT_DIS fields
Table 132. sep_efuse_map.TRANSIENT_RMA_EN fields
Table 133. sep_efuse_map.SIP_DIS fields
Table 134. sep_efuse_map.SYS_DIS fields
Table 135. sep_efuse_map.RMA_SIP_TOKEN_DIGEST fields
Table 136. sep_efuse_map.RMA_CHIPLET_TOKEN_DIGEST fields
Table 137. sep_efuse_map.CLASS_KEY fields
Table 138. sep_efuse_map.CHIPLET_PUBK_REVOKE fields
Table 139. sep_efuse_map.BL1_VERSION fields
Table 140. sep_efuse_map.BL2_VERSION fields
Table 141. sep_efuse_map.CHIPLET_UID fields
Table 142. sep_efuse_map.SIP_PUBK_HASH0 fields

Table 143. sep_efuse_map.SIP_UID fields

Table 144. sep_efuse_map.SYS_PUBK_HASH fields

Table 145. sep_efuse_map.SYS_UID fields

Table 146. sep_efuse_map.STATUS_RPT fields

Table 147. sep_efuse_map.ROM_CTL fields

Table 148. sep_efuse_map.SYSCLK_FREQ_MHZ fields

Table 149. sep_efuse_map.CHIPLET_PUBK_HASH0 fields

Table 150. sep_efuse_map.CHIPLET_PUBK_HASH1 fields

Table 151. sep_efuse_map.REQUIRED_SIGNERS fields

Table 152. sep_efuse_map.REQUIRED_ALGS fields

Table 153. sep_efuse_map.CHIPLET_PUBK_PQC_HASH0 fields

Table 154. sep_efuse_map.CHIPLET_PUBK_PQC_HASH1 fields

Table 155. sep_efuse_map.SIP_PUBK_PQC_HASH0 fields

Table 156. sep_efuse_map.SYS_PUBK_PQC_HASH fields

Table 157. sep_efuse_map.SIP_PUBK_HASH1 fields

Table 158. sep_efuse_map.SIP_PUBK_PQC_HASH1 fields

Table 159. sep_efuse_map.SEP_CHIPLET_ID fields

Table 160. sep_efuse_map.SEP_SIP_ID fields

Table 161. sep_efuse_map.SEP_SYS_ID fields

Table 162. sep_efuse_map.spare0 fields

Table 163. sep_efuse_map.spare1 fields

Table 164. sep_efuse_map.spare2 fields

Table 165. sep_efuse_map.spare3 fields

Table 166. sep_efuse_map.spare4 fields

Table 167. sep_efuse_map.spare5 fields

Table 168. sep_efuse_map.spare6 fields

Table 169. sep_efuse_map.spare7 fields

Table 170. sep_efuse_map.spare8 fields

Table 171. Token roles and expected digests

Table 172. Token submission registers

Table 173. Token result encodings

Table 174. SEP lifecycle state encodings

Table 175. SEP feature-disable vector

Table 176. SEP disable-vector applicability

Table 177. Per-LC-state feature control profile

Table 178. SEP demotion controls

Table 179. SEP security asset classes

Table 180. SEP scan-chain gates

Table 181. DTP access cases

Table 182. DTP path feature gates

Table 183. SEP Lifecycle Control Address Map register list

Table 184. sep_lifecycle_ctrl.FEAT_CTRL fields

Table 185. sep_lifecycle_ctrl.DEMOTE_1 fields

Table 186. sep_lifecycle_ctrl.DEMOTE_2 fields

Table 187. Secure test-mode sampling in the SEP wrapper

Table 188. Required secure-test protections

Table 189. SEP crypto alert and fault routing

Table 190. Key Manager key delivery per accelerator

Table 191. SEP entropy sink paths

Table 192. SEP random-data stream contract

Table 193. TRNG reset effects outside the internal domain

Table 194. DRBG external interfaces

Table 195. DRBG lane adapter forwarded accesses

Table 196. Cryptographically Secure Random Number Generator register list

Table 197. csrng.INTR_STATE fields

Table 198. csrng.INTR_ENABLE fields

Table 199. csrng.INTR_TEST fields

Table 200. csrng.ALERT_TEST fields

Table 201. csrng.REGWEN fields

Table 202. csrng.CTRL fields

Table 203. csrng.CMD_REQ fields

Table 204. csrng.RESEED_INTERVAL fields

Table 205. csrng.RESEED_COUNTER_0 fields

Table 206. csrng.RESEED_COUNTER_1 fields

Table 207. csrng.RESEED_COUNTER_2 fields

Table 208. csrng.SW_CMD_STS fields

Table 209. csrng.GENBITS_VLD fields

Table 210. csrng.GENBITS fields

Table 211. csrng.INT_STATE_READ_ENABLE fields

Table 212. csrng.INT_STATE_READ_ENABLE_REGWEN fields

Table 213. csrng.INT_STATE_NUM fields

Table 214. csrng.INT_STATE_VAL fields

Table 215. csrng.FIPS_FORCE fields
Table 216. csrng.HW_EXC_STS fields
Table 217. csrng.RECOV_ALERT_STS fields
Table 218. csrng.ERR_CODE fields
Table 219. csrng.ERR_CODE_TEST fields
Table 220. csrng.MAIN_SM_STATE fields
Table 221. Entropy Distribution Network register list
Table 222. edn.INTR_STATE fields
Table 223. edn.INTR_ENABLE fields
Table 224. edn.INTR_TEST fields
Table 225. edn.ALERT_TEST fields
Table 226. edn.REGWEN fields
Table 227. edn.CTRL fields
Table 228. edn.BOOT_INS_CMD fields
Table 229. edn.BOOT_GEN_CMD fields
Table 230. edn.SW_CMD_REQ fields
Table 231. edn.SW_CMD_STS fields
Table 232. edn.HW_CMD_STS fields
Table 233. edn.RESEED_CMD fields
Table 234. edn.GENERATE_CMD fields
Table 235. edn.MAX_NUM_REQS_BETWEEN_RESEEDS fields
Table 236. edn.RECOV_ALERT_STS fields
Table 237. edn.ERR_CODE fields
Table 238. edn.ERR_CODE_TEST fields
Table 239. edn.MAIN_SM_STATE fields
Table 240. Noise ring-oscillator lengths per lane
Table 241. Readable FIFOs
Table 242. Entropy Source Address Map register list
Table 243. entropy_source.COMPONENT_ID fields
Table 244. entropy_source.CTRL fields
Table 245. entropy_source.DEBUG_CTRL fields
Table 246. entropy_source.INTR_STATUS fields
Table 247. entropy_source.INTR_ENABLE fields
Table 248. entropy_source.INTR_TEST fields
Table 249. entropy_source.SHA256_STATUS fields
Table 250. entropy_source.FIFO_CTRL fields

Table 251. entropy_source.FIFO_STATUS fields
Table 252. entropy_source.FIFO_RDATA fields
Table 253. entropy_source.HEALTH_TEST_CTRL fields
Table 254. entropy_source.HEALTH_TEST_WINDOW_SIZE fields
Table 255. entropy_source.MARKOV_TEST_PROB_THRESHOLDS fields
Table 256. entropy_source.HEALTH_TEST_STATUS fields
Table 257. entropy_source.REPETITION_TEST_COUNT fields
Table 258. entropy_source.APT_PATTERN_COUNT_1BIT fields
Table 259. entropy_source.APT_PATTERN_COUNT_2BIT fields
Table 260. entropy_source.APT_PROPORTION_1BIT fields
Table 261. entropy_source.APT_PROPORTION_LO fields
Table 262. entropy_source.MARKOV_TEST_COUNTS_0 fields
Table 263. entropy_source.RING_OSC_ENABLE fields
Table 264. entropy_source.RING_OSC_TUNE fields
Table 265. entropy_source.RING_OSC_CTRL fields
Table 266. entropy_source.DECORRELATOR_CTRL fields
Table 267. entropy_source.DECORRELATOR_MASK fields
Table 268. entropy_source.MAIN_SM_STATUS fields
Table 269. entropy_source.GENERATOR_0_HEALTH_STATUS fields
Table 270. entropy_source.GENERATOR_1_HEALTH_STATUS fields
Table 271. entropy_source.GENERATOR_2_HEALTH_STATUS fields
Table 272. entropy_source.GENERATOR_3_HEALTH_STATUS fields
Table 273. entropy_source.GENERATOR_4_HEALTH_STATUS fields
Table 274. entropy_source.GENERATOR_5_HEALTH_STATUS fields
Table 275. entropy_source.GENERATOR_6_HEALTH_STATUS fields
Table 276. entropy_source.GENERATOR_7_HEALTH_STATUS fields
Table 277. entropy_source.GENERATOR_8_HEALTH_STATUS fields
Table 278. entropy_source.GENERATOR_9_HEALTH_STATUS fields
Table 279. entropy_source.GENERATOR_10_HEALTH_STATUS fields
Table 280. entropy_source.GENERATOR_11_HEALTH_STATUS fields
Table 281. entropy_source.GENERATOR_0_SAMPLE_CLK_CONFIG fields
Table 282. entropy_source.GENERATOR_1_SAMPLE_CLK_CONFIG fields
Table 283. entropy_source.GENERATOR_2_SAMPLE_CLK_CONFIG fields
Table 284. entropy_source.GENERATOR_3_SAMPLE_CLK_CONFIG fields
Table 285. entropy_source.GENERATOR_4_SAMPLE_CLK_CONFIG fields
Table 286. entropy_source.GENERATOR_5_SAMPLE_CLK_CONFIG fields

Table 287. entropy_source.GENERATOR_6_SAMPLE_CLK_CONFIG fields
Table 288. entropy_source.GENERATOR_7_SAMPLE_CLK_CONFIG fields
Table 289. entropy_source.GENERATOR_8_SAMPLE_CLK_CONFIG fields
Table 290. entropy_source.GENERATOR_9_SAMPLE_CLK_CONFIG fields
Table 291. entropy_source.GENERATOR_10_SAMPLE_CLK_CONFIG fields
Table 292. entropy_source.GENERATOR_11_SAMPLE_CLK_CONFIG fields
Table 293. entropy_source.HT_WATERMARK_NUM fields
Table 294. entropy_source.HT_WATERMARK fields
Table 295. entropy_source.REPCNT_TOTAL_FAILS fields
Table 296. entropy_source.APT_HI_TOTAL_FAILS fields
Table 297. entropy_source.APT_LO_TOTAL_FAILS fields
Table 298. entropy_source.MARKOV_HI_TOTAL_FAILS fields
Table 299. entropy_source.MARKOV_LO_TOTAL_FAILS fields
Table 300. entropy_source.ALERT_SUMMARY_FAIL_COUNTS fields
Table 301. entropy_source.ALERT_FAIL_COUNTS fields
Table 302. entropy_source.FIPS_LOCK fields
Table 303. entropy_source.ALERT_THRESHOLD fields
Table 304. entropy_source.MIN_ENTROPY_H fields
Table 305. entropy_source.RECOMMENDED_THRESHOLDS fields
Table 306. entropy_source.BIW_OBS_CTRL fields
Table 307. entropy_source.BIW_OBS_STATUS fields
Table 308. entropy_source.BIW_OBS_RDATA fields
Table 309. entropy_source.NOISE_OBS_CTRL fields
Table 310. entropy_source.NOISE_OBS_STATUS fields
Table 311. entropy_source.NOISE_OBS_RDATA fields
Table 312. SEP Entropy Pool register list
Table 313. sep_entropy_pool.STATUS fields
Table 314. sep_entropy_pool.IRQ_CAUSE fields
Table 315. sep_entropy_pool.DATA fields
Table 316. AES Accelerator register list
Table 317. aes.ALERT_TEST fields
Table 318. aes.KEY_SHARE0[8] fields
Table 319. aes.KEY_SHARE1[8] fields
Table 320. aes.IV[4] fields
Table 321. aes.DATA_IN[4] fields
Table 322. aes.DATA_OUT[4] fields

Table 323. aes.CTRL_SHADOWED fields
Table 324. aes.CTRL_AUX_SHADOWED fields
Table 325. aes.CTRL_AUX_REGWEN fields
Table 326. aes.TRIGGER fields
Table 327. aes.STATUS fields
Table 328. aes.CTRL_GCM_SHADOWED fields
Table 329. HMAC Accelerator register list
Table 330. hmac.INTR_STATE fields
Table 331. hmac.INTR_ENABLE fields
Table 332. hmac.INTR_TEST fields
Table 333. hmac.ALERT_TEST fields
Table 334. hmac.CFG fields
Table 335. hmac.CMD fields
Table 336. hmac.STATUS fields
Table 337. hmac.ERR_CODE fields
Table 338. hmac.WIPE_SECRET fields
Table 339. hmac.KEY[32] fields
Table 340. hmac.DIGEST[16] fields
Table 341. hmac.MSG_LENGTH_LOWER fields
Table 342. hmac.MSG_LENGTH_UPPER fields
Table 343. KMAC Accelerator register list
Table 344. kmac.INTR_STATE fields
Table 345. kmac.INTR_ENABLE fields
Table 346. kmac.INTR_TEST fields
Table 347. kmac.ALERT_TEST fields
Table 348. kmac.CFG_REGWEN fields
Table 349. kmac.CFG_SHADOWED fields
Table 350. kmac.CMD fields
Table 351. kmac.STATUS fields
Table 352. kmac.ENTROPY_PERIOD fields
Table 353. kmac.ENTROPY_REFRESH_HASH_CNT fields
Table 354. kmac.ENTROPY_REFRESH_THRESHOLD_SHADOWED fields
Table 355. kmac.ENTROPY_SEED fields
Table 356. kmac.KEY_SHARE0[16] fields
Table 357. kmac.KEY_SHARE1[16] fields
Table 358. kmac.KEY_LEN fields

Table 359. kmac.PREFIX[11] fields

Table 360. kmac.ERR_CODE fields

Table 361. OpenTitan Big Number Accelerator register list

Table 362. otbn.INTR_STATE fields

Table 363. otbn.INTR_ENABLE fields

Table 364. otbn.INTR_TEST fields

Table 365. otbn.ALERT_TEST fields

Table 366. otbn.CMD fields

Table 367. otbn.CTRL fields

Table 368. otbn.STATUS fields

Table 369. otbn.ERR_BITS fields

Table 370. otbn.FATAL_ALERT_CAUSE fields

Table 371. otbn.INSN_CNT fields

Table 372. otbn.LOAD_CHECKSUM fields

Table 373. Key Manager reset inputs

Table 374. Key Manager reset-domain assignment

Table 375. Key Manager message with payload

Table 376. Key Manager header-only message

Table 377. Adams Bridge seed sideload inputs

Table 378. KM CPU interrupt routing

Table 379. KPV per-slot controls

Table 380. Key Manager VERSION fields

Table 381. Key Manager interrupt sources

Table 382. Key Manager executable regions

Table 383. DRBG sampler register behavior

Table 384. Key Manager DRBG sampler states

Table 385. Key Manager OTP readout registers

Table 386. CRC Accelerator Instructions

Table 387. Key Manager warm-persistent SRAM contents

Table 388. SRAM scrambler S-box

Table 389. SRAM scrambler permutation for bits 31 to 16

Table 390. SRAM scrambler permutation for bits 15 to 0

Table 391. SRAM scrambler address expansion

Table 392. SRAM scrambler address lanes

Table 393. SRAM scrambler module variants

Table 394. Key Manager CPU Memory Map

Table 395. Key Manager Top-Level Address Map register list
Table 396. key_manager.KM_WRITE_DATA fields
Table 397. key_manager.KM_WRITE_SEPARATOR fields
Table 398. key_manager.KM_READ_DATA fields
Table 399. key_manager.KM_STATUS fields
Table 400. key_manager.KM_IRQ_STATUS fields
Table 401. key_manager.KM_IRQ_ENABLE fields
Table 402. key_manager.KM_CTRL fields
Table 403. key_manager.LOCKS fields
Table 404. key_manager.LOCKS_SPARE fields
Table 405. key_manager.LC_STATE fields
Table 406. key_manager.SBOOT_DIS fields
Table 407. key_manager.TRANSIENT_RMA_EN fields
Table 408. key_manager.SIP_DIS fields
Table 409. key_manager.SYS_DIS fields
Table 410. key_manager.RMA_SIP_TOKEN_DIGEST fields
Table 411. key_manager.RMA_CHIPLET_TOKEN_DIGEST fields
Table 412. key_manager.CLASS_KEY fields
Table 413. key_manager.CHIPLET_PUBK_REVOKE fields
Table 414. key_manager.BL1_VERSION fields
Table 415. key_manager.BL2_VERSION fields
Table 416. key_manager.CHIPLET_UID fields
Table 417. key_manager.SIP_PUBK_HASH0 fields
Table 418. key_manager.SIP_UID fields
Table 419. key_manager.SYS_PUBK_HASH fields
Table 420. key_manager.SYS_UID fields
Table 421. key_manager.STATUS_RPT fields
Table 422. key_manager.ROM_CTL fields
Table 423. key_manager.SYSCLK_FREQ_MHZ fields
Table 424. key_manager.CHIPLET_PUBK_HASH0 fields
Table 425. key_manager.CHIPLET_PUBK_HASH1 fields
Table 426. key_manager.REQUIRED_SIGNERS fields
Table 427. key_manager.REQUIRED_ALGS fields
Table 428. key_manager.CHIPLET_PUBK_PQC_HASH0 fields
Table 429. key_manager.CHIPLET_PUBK_PQC_HASH1 fields
Table 430. key_manager.SIP_PUBK_PQC_HASH0 fields

Table 431. key_manager.SYS_PUBK_PQC_HASH fields

Table 432. key_manager.SIP_PUBK_HASH1 fields

Table 433. key_manager.SIP_PUBK_PQC_HASH1 fields

Table 434. key_manager.SEP_CHIPLET_ID fields

Table 435. key_manager.SEP_SIP_ID fields

Table 436. key_manager.SEP_SYS_ID fields

Table 437. key_manager.spare0 fields

Table 438. key_manager.spare1 fields

Table 439. key_manager.spare2 fields

Table 440. key_manager.spare3 fields

Table 441. key_manager.spare4 fields

Table 442. key_manager.spare5 fields

Table 443. key_manager.spare6 fields

Table 444. key_manager.spare7 fields

Table 445. key_manager.spare8 fields

Table 446. key_manager.EFUSE_INTERFACE_CTRL_STATUS fields

Table 447. key_manager.EFUSE_PROGRAM_CTRL fields

Table 448. key_manager.EFUSE_READ_CTRL fields

Table 449. key_manager.EFUSE_PROGRAM_INTERFACE_READ_DATA fields

Table 450. key_manager.EFUSE_READ_INTERFACE_READ_DATA fields

Table 451. key_manager.EFUSE_READ_REQ_TIMEOUT fields

Table 452. key_manager.EFUSE_PROGRAM_REQ_TIMEOUT fields

Table 453. key_manager.RMA_SIP_TOKEN_I[8] fields

Table 454. key_manager.RMA_CHIPLET_TOKEN_I[8] fields

Table 455. key_manager.SEC_DISABLE_TOKEN_I[8] fields

Table 456. key_manager.TOKEN_EOP fields

Table 457. key_manager.RMA_SIP_TOKEN_MATCH fields

Table 458. key_manager.RMA_CHIPLET_TOKEN_MATCH fields

Table 459. key_manager.SEC_DISABLE_TOKEN_MATCH fields

Table 460. key_manager.TOKEN_MATCH_FAULT fields

Table 461. key_manager.KEY_ENTRY[64].WORD[16] fields

Table 462. key_manager.CTRL[64] fields

Table 463. key_manager.KPV_SCRAMBLER_KEY fields

Table 464. key_manager.KPV_SCRAMBLER_CTRL fields

Table 465. key_manager.VERSION fields

Table 466. key_manager.CTRL fields

Table 467. key_manager.SOFT_RST_CODE fields
Table 468. key_manager.kmcsr.IRQ_STATUS fields
Table 469. key_manager.kmcsr.IRQ_ENABLE fields
Table 470. key_manager.SCRAMBLER_KEY fields
Table 471. key_manager.SCRAMBLER_CTRL fields
Table 472. key_manager.SRAM_LOCK fields
Table 473. key_manager.IRQ_SET fields
Table 474. key_manager.SRAM_WRITE_LOCK_VIOLATION fields
Table 475. key_manager.RECOVERABLE_ERR fields
Table 476. key_manager.BOOT_STATUS fields
Table 477. key_manager.OTP_LIFE_CYCLE fields
Table 478. key_manager.OTP_DEMOTION_STATE fields
Table 479. key_manager.SRAM_EXEC_MODE fields
Table 480. key_manager.IRQ_ENTRY_ADDR fields
Table 481. key_manager.IRQ_ENTRY_LOCK fields
Table 482. key_manager.VUART_TX fields
Table 483. key_manager.VUART_RX fields
Table 484. key_manager.VUART_STATUS fields
Table 485. key_manager.TB_RESULT fields
Table 486. key_manager.TB_SIGNATURE fields
Table 487. key_manager.TB_ERRCODE fields
Table 488. key_manager.TB_SUBTEST fields
Table 489. key_manager.TB_CMD fields
Table 490. key_manager.TB_CMD_ARG fields
Table 491. key_manager.TB_CMD_STATUS fields
Table 492. key_manager.TB_CMD_RESULT fields
Table 493. key_manager.DEBUG fields
Table 494. key_manager.OTP_CHIPLET_UID_VAL_0 fields
Table 495. key_manager.OTP_CHIPLET_UID_VAL_1 fields
Table 496. key_manager.OTP_CHIPLET_UID_VAL_2 fields
Table 497. key_manager.OTP_CHIPLET_UID_VAL_3 fields
Table 498. key_manager.OTP_CHIPLET_UID_VAL_4 fields
Table 499. key_manager.OTP_CHIPLET_UID_VAL_5 fields
Table 500. key_manager.OTP_CHIPLET_UID_VAL_6 fields
Table 501. key_manager.OTP_CHIPLET_UID_VAL_7 fields
Table 502. key_manager.OTP_CHIPLET_UID_CPL_0 fields

Table 503. key_manager.OTP_CHIPLET_UID_CPL_1 fields
Table 504. key_manager.OTP_CHIPLET_UID_CPL_2 fields
Table 505. key_manager.OTP_CHIPLET_UID_CPL_3 fields
Table 506. key_manager.OTP_CHIPLET_UID_CPL_4 fields
Table 507. key_manager.OTP_CHIPLET_UID_CPL_5 fields
Table 508. key_manager.OTP_CHIPLET_UID_CPL_6 fields
Table 509. key_manager.OTP_CHIPLET_UID_CPL_7 fields
Table 510. key_manager.OTP_SIP_UID_VAL_0 fields
Table 511. key_manager.OTP_SIP_UID_VAL_1 fields
Table 512. key_manager.OTP_SIP_UID_VAL_2 fields
Table 513. key_manager.OTP_SIP_UID_VAL_3 fields
Table 514. key_manager.OTP_SIP_UID_VAL_4 fields
Table 515. key_manager.OTP_SIP_UID_VAL_5 fields
Table 516. key_manager.OTP_SIP_UID_VAL_6 fields
Table 517. key_manager.OTP_SIP_UID_VAL_7 fields
Table 518. key_manager.OTP_SIP_UID_CPL_0 fields
Table 519. key_manager.OTP_SIP_UID_CPL_1 fields
Table 520. key_manager.OTP_SIP_UID_CPL_2 fields
Table 521. key_manager.OTP_SIP_UID_CPL_3 fields
Table 522. key_manager.OTP_SIP_UID_CPL_4 fields
Table 523. key_manager.OTP_SIP_UID_CPL_5 fields
Table 524. key_manager.OTP_SIP_UID_CPL_6 fields
Table 525. key_manager.OTP_SIP_UID_CPL_7 fields
Table 526. key_manager.OTP_SYS_UID_VAL_0 fields
Table 527. key_manager.OTP_SYS_UID_VAL_1 fields
Table 528. key_manager.OTP_SYS_UID_VAL_2 fields
Table 529. key_manager.OTP_SYS_UID_VAL_3 fields
Table 530. key_manager.OTP_SYS_UID_VAL_4 fields
Table 531. key_manager.OTP_SYS_UID_VAL_5 fields
Table 532. key_manager.OTP_SYS_UID_VAL_6 fields
Table 533. key_manager.OTP_SYS_UID_VAL_7 fields
Table 534. key_manager.OTP_SYS_UID_CPL_0 fields
Table 535. key_manager.OTP_SYS_UID_CPL_1 fields
Table 536. key_manager.OTP_SYS_UID_CPL_2 fields
Table 537. key_manager.OTP_SYS_UID_CPL_3 fields
Table 538. key_manager.OTP_SYS_UID_CPL_4 fields

Table 539. key_manager.OTP_SYS_UID_CPL_5 fields
Table 540. key_manager.OTP_SYS_UID_CPL_6 fields
Table 541. key_manager.OTP_SYS_UID_CPL_7 fields
Table 542. key_manager.OTP_CLASS_KEY_VAL_0 fields
Table 543. key_manager.OTP_CLASS_KEY_VAL_1 fields
Table 544. key_manager.OTP_CLASS_KEY_VAL_2 fields
Table 545. key_manager.OTP_CLASS_KEY_VAL_3 fields
Table 546. key_manager.OTP_CLASS_KEY_VAL_4 fields
Table 547. key_manager.OTP_CLASS_KEY_VAL_5 fields
Table 548. key_manager.OTP_CLASS_KEY_VAL_6 fields
Table 549. key_manager.OTP_CLASS_KEY_VAL_7 fields
Table 550. key_manager.OTP_CLASS_KEY_CPL_0 fields
Table 551. key_manager.OTP_CLASS_KEY_CPL_1 fields
Table 552. key_manager.OTP_CLASS_KEY_CPL_2 fields
Table 553. key_manager.OTP_CLASS_KEY_CPL_3 fields
Table 554. key_manager.OTP_CLASS_KEY_CPL_4 fields
Table 555. key_manager.OTP_CLASS_KEY_CPL_5 fields
Table 556. key_manager.OTP_CLASS_KEY_CPL_6 fields
Table 557. key_manager.OTP_CLASS_KEY_CPL_7 fields
Table 558. key_manager.OTP_READ_LOCK fields
Table 559. key_manager.OTP_CHANGE_STATUS fields
Table 560. key_manager.OTP_READ_LOCK_COLD fields
Table 561. key_manager.OTP_SEP_CHIPLET_ID_VAL_0 fields
Table 562. key_manager.OTP_SEP_CHIPLET_ID_VAL_1 fields
Table 563. key_manager.OTP_SEP_CHIPLET_ID_VAL_2 fields
Table 564. key_manager.OTP_SEP_CHIPLET_ID_VAL_3 fields
Table 565. key_manager.OTP_SEP_CHIPLET_ID_VAL_4 fields
Table 566. key_manager.OTP_SEP_CHIPLET_ID_VAL_5 fields
Table 567. key_manager.OTP_SEP_CHIPLET_ID_VAL_6 fields
Table 568. key_manager.OTP_SEP_CHIPLET_ID_VAL_7 fields
Table 569. key_manager.OTP_SEP_CHIPLET_ID_CPL_0 fields
Table 570. key_manager.OTP_SEP_CHIPLET_ID_CPL_1 fields
Table 571. key_manager.OTP_SEP_CHIPLET_ID_CPL_2 fields
Table 572. key_manager.OTP_SEP_CHIPLET_ID_CPL_3 fields
Table 573. key_manager.OTP_SEP_CHIPLET_ID_CPL_4 fields
Table 574. key_manager.OTP_SEP_CHIPLET_ID_CPL_5 fields

Table 575. key_manager.OTP_SEP_CHIPLET_ID_CPL_6 fields
Table 576. key_manager.OTP_SEP_CHIPLET_ID_CPL_7 fields
Table 577. key_manager.OTP_SEP_SIP_ID_VAL_0 fields
Table 578. key_manager.OTP_SEP_SIP_ID_VAL_1 fields
Table 579. key_manager.OTP_SEP_SIP_ID_VAL_2 fields
Table 580. key_manager.OTP_SEP_SIP_ID_VAL_3 fields
Table 581. key_manager.OTP_SEP_SIP_ID_VAL_4 fields
Table 582. key_manager.OTP_SEP_SIP_ID_VAL_5 fields
Table 583. key_manager.OTP_SEP_SIP_ID_VAL_6 fields
Table 584. key_manager.OTP_SEP_SIP_ID_VAL_7 fields
Table 585. key_manager.OTP_SEP_SIP_ID_CPL_0 fields
Table 586. key_manager.OTP_SEP_SIP_ID_CPL_1 fields
Table 587. key_manager.OTP_SEP_SIP_ID_CPL_2 fields
Table 588. key_manager.OTP_SEP_SIP_ID_CPL_3 fields
Table 589. key_manager.OTP_SEP_SIP_ID_CPL_4 fields
Table 590. key_manager.OTP_SEP_SIP_ID_CPL_5 fields
Table 591. key_manager.OTP_SEP_SIP_ID_CPL_6 fields
Table 592. key_manager.OTP_SEP_SIP_ID_CPL_7 fields
Table 593. key_manager.OTP_SEP_SYS_ID_VAL_0 fields
Table 594. key_manager.OTP_SEP_SYS_ID_VAL_1 fields
Table 595. key_manager.OTP_SEP_SYS_ID_VAL_2 fields
Table 596. key_manager.OTP_SEP_SYS_ID_VAL_3 fields
Table 597. key_manager.OTP_SEP_SYS_ID_VAL_4 fields
Table 598. key_manager.OTP_SEP_SYS_ID_VAL_5 fields
Table 599. key_manager.OTP_SEP_SYS_ID_VAL_6 fields
Table 600. key_manager.OTP_SEP_SYS_ID_VAL_7 fields
Table 601. key_manager.OTP_SEP_SYS_ID_CPL_0 fields
Table 602. key_manager.OTP_SEP_SYS_ID_CPL_1 fields
Table 603. key_manager.OTP_SEP_SYS_ID_CPL_2 fields
Table 604. key_manager.OTP_SEP_SYS_ID_CPL_3 fields
Table 605. key_manager.OTP_SEP_SYS_ID_CPL_4 fields
Table 606. key_manager.OTP_SEP_SYS_ID_CPL_5 fields
Table 607. key_manager.OTP_SEP_SYS_ID_CPL_6 fields
Table 608. key_manager.OTP_SEP_SYS_ID_CPL_7 fields
Table 609. key_manager.DATA fields
Table 610. key_manager.CFG fields

Table 611. key_manager.STATUS fields

Table 612. key_manager.PREFETCH_DATA fields

Table 613. key_manager.KEY_SHARE0[12] fields

Table 614. key_manager.KEY_SHARE1[12] fields

Table 615. key_manager.otbn_wrapper_key.KEY_CTRL fields

Table 616. key_manager.aes_wrapper_key.KEY_SHARE0[8] fields

Table 617. key_manager.aes_wrapper_key.KEY_SHARE1[8] fields

Table 618. key_manager.aes_wrapper_key.KEY_CTRL fields

Table 619. key_manager.kmac_wrapper_key.KEY_SHARE0[8] fields

Table 620. key_manager.kmac_wrapper_key.KEY_SHARE1[8] fields

Table 621. key_manager.kmac_wrapper_key.KEY_CTRL fields

Table 622. key_manager.hmac_wrapper_key.KEY_SHARE0[8] fields

Table 623. key_manager.hmac_wrapper_key.KEY_SHARE1[8] fields

Table 624. key_manager.hmac_wrapper_key.KEY_CTRL fields

Table 625. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] fields

Table 626. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] fields

Table 627. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_CTRL fields

Table 628. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] fields

Table 629. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] fields

Table 630. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL fields

Table 631. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] fields

Table 632. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] fields

Table 633. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL fields

Table 634. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] fields

Table 635. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] fields

Table 636. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_CTRL fields

Table 637. key_manager.KEY[8] fields

Table 638. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL fields

Table 639. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_STATUS fields

Table 640. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_ENABLE fields

Table 641. Key Manager Control and Status Registers register list

Table 642. km_csr.VERSION fields

Table 643. km_csr.CTRL fields

Table 644. km_csr.SOFT_RST_CODE fields

Table 645. km_csr.IRQ_STATUS fields

Table 646. km_csr.IRQ_ENABLE fields

Table 647. km_csr.SCRAMBLER_KEY fields

Table 648. km_csr.SCRAMBLER_CTRL fields

Table 649. km_csr.SRAM_LOCK fields

Table 650. km_csr.IRQ_SET fields

Table 651. km_csr.SRAM_WRITE_LOCK_VIOLATION fields

Table 652. km_csr.RECOVERABLE_ERR fields

Table 653. km_csr.BOOT_STATUS fields

Table 654. km_csr.OTP_LIFE_CYCLE fields

Table 655. km_csr.OTP_DEMOTION_STATE fields

Table 656. km_csr.SRAM_EXEC_MODE fields

Table 657. km_csr.IRQ_ENTRY_ADDR fields

Table 658. km_csr.IRQ_ENTRY_LOCK fields

Table 659. km_csr.VUART_TX fields

Table 660. km_csr.VUART_RX fields

Table 661. km_csr.VUART_STATUS fields

Table 662. km_csr.TB_RESULT fields

Table 663. km_csr.TB_SIGNATURE fields

Table 664. km_csr.TB_ERRCODE fields

Table 665. km_csr.TB_SUBTEST fields

Table 666. km_csr.TB_CMD fields

Table 667. km_csr.TB_CMD_ARG fields

Table 668. km_csr.TB_CMD_STATUS fields

Table 669. km_csr.TB_CMD_RESULT fields

Table 670. km_csr.DEBUG fields

Table 671. km_csr.OTP_CHIPLET_UID_VAL_0 fields

Table 672. km_csr.OTP_CHIPLET_UID_VAL_1 fields

Table 673. km_csr.OTP_CHIPLET_UID_VAL_2 fields

Table 674. km_csr.OTP_CHIPLET_UID_VAL_3 fields

Table 675. km_csr.OTP_CHIPLET_UID_VAL_4 fields

Table 676. km_csr.OTP_CHIPLET_UID_VAL_5 fields

Table 677. km_csr.OTP_CHIPLET_UID_VAL_6 fields

Table 678. km_csr.OTP_CHIPLET_UID_VAL_7 fields

Table 679. km_csr.OTP_CHIPLET_UID_CPL_0 fields

Table 680. km_csr.OTP_CHIPLET_UID_CPL_1 fields

Table 681. km_csr.OTP_CHIPLET_UID_CPL_2 fields

Table 682. km_csr.OTP_CHIPLET_UID_CPL_3 fields

Table 683. km_csr.OTP_CHIPLET_UID_CPL_4 fields
Table 684. km_csr.OTP_CHIPLET_UID_CPL_5 fields
Table 685. km_csr.OTP_CHIPLET_UID_CPL_6 fields
Table 686. km_csr.OTP_CHIPLET_UID_CPL_7 fields
Table 687. km_csr.OTP_SIP_UID_VAL_0 fields
Table 688. km_csr.OTP_SIP_UID_VAL_1 fields
Table 689. km_csr.OTP_SIP_UID_VAL_2 fields
Table 690. km_csr.OTP_SIP_UID_VAL_3 fields
Table 691. km_csr.OTP_SIP_UID_VAL_4 fields
Table 692. km_csr.OTP_SIP_UID_VAL_5 fields
Table 693. km_csr.OTP_SIP_UID_VAL_6 fields
Table 694. km_csr.OTP_SIP_UID_VAL_7 fields
Table 695. km_csr.OTP_SIP_UID_CPL_0 fields
Table 696. km_csr.OTP_SIP_UID_CPL_1 fields
Table 697. km_csr.OTP_SIP_UID_CPL_2 fields
Table 698. km_csr.OTP_SIP_UID_CPL_3 fields
Table 699. km_csr.OTP_SIP_UID_CPL_4 fields
Table 700. km_csr.OTP_SIP_UID_CPL_5 fields
Table 701. km_csr.OTP_SIP_UID_CPL_6 fields
Table 702. km_csr.OTP_SIP_UID_CPL_7 fields
Table 703. km_csr.OTP_SYS_UID_VAL_0 fields
Table 704. km_csr.OTP_SYS_UID_VAL_1 fields
Table 705. km_csr.OTP_SYS_UID_VAL_2 fields
Table 706. km_csr.OTP_SYS_UID_VAL_3 fields
Table 707. km_csr.OTP_SYS_UID_VAL_4 fields
Table 708. km_csr.OTP_SYS_UID_VAL_5 fields
Table 709. km_csr.OTP_SYS_UID_VAL_6 fields
Table 710. km_csr.OTP_SYS_UID_VAL_7 fields
Table 711. km_csr.OTP_SYS_UID_CPL_0 fields
Table 712. km_csr.OTP_SYS_UID_CPL_1 fields
Table 713. km_csr.OTP_SYS_UID_CPL_2 fields
Table 714. km_csr.OTP_SYS_UID_CPL_3 fields
Table 715. km_csr.OTP_SYS_UID_CPL_4 fields
Table 716. km_csr.OTP_SYS_UID_CPL_5 fields
Table 717. km_csr.OTP_SYS_UID_CPL_6 fields
Table 718. km_csr.OTP_SYS_UID_CPL_7 fields

Table 719. km_csr.OTP_CLASS_KEY_VAL_0 fields
Table 720. km_csr.OTP_CLASS_KEY_VAL_1 fields
Table 721. km_csr.OTP_CLASS_KEY_VAL_2 fields
Table 722. km_csr.OTP_CLASS_KEY_VAL_3 fields
Table 723. km_csr.OTP_CLASS_KEY_VAL_4 fields
Table 724. km_csr.OTP_CLASS_KEY_VAL_5 fields
Table 725. km_csr.OTP_CLASS_KEY_VAL_6 fields
Table 726. km_csr.OTP_CLASS_KEY_VAL_7 fields
Table 727. km_csr.OTP_CLASS_KEY_CPL_0 fields
Table 728. km_csr.OTP_CLASS_KEY_CPL_1 fields
Table 729. km_csr.OTP_CLASS_KEY_CPL_2 fields
Table 730. km_csr.OTP_CLASS_KEY_CPL_3 fields
Table 731. km_csr.OTP_CLASS_KEY_CPL_4 fields
Table 732. km_csr.OTP_CLASS_KEY_CPL_5 fields
Table 733. km_csr.OTP_CLASS_KEY_CPL_6 fields
Table 734. km_csr.OTP_CLASS_KEY_CPL_7 fields
Table 735. km_csr.OTP_READ_LOCK fields
Table 736. km_csr.OTP_CHANGE_STATUS fields
Table 737. km_csr.OTP_READ_LOCK_COLD fields
Table 738. km_csr.OTP_SEP_CHIPLET_ID_VAL_0 fields
Table 739. km_csr.OTP_SEP_CHIPLET_ID_VAL_1 fields
Table 740. km_csr.OTP_SEP_CHIPLET_ID_VAL_2 fields
Table 741. km_csr.OTP_SEP_CHIPLET_ID_VAL_3 fields
Table 742. km_csr.OTP_SEP_CHIPLET_ID_VAL_4 fields
Table 743. km_csr.OTP_SEP_CHIPLET_ID_VAL_5 fields
Table 744. km_csr.OTP_SEP_CHIPLET_ID_VAL_6 fields
Table 745. km_csr.OTP_SEP_CHIPLET_ID_VAL_7 fields
Table 746. km_csr.OTP_SEP_CHIPLET_ID_CPL_0 fields
Table 747. km_csr.OTP_SEP_CHIPLET_ID_CPL_1 fields
Table 748. km_csr.OTP_SEP_CHIPLET_ID_CPL_2 fields
Table 749. km_csr.OTP_SEP_CHIPLET_ID_CPL_3 fields
Table 750. km_csr.OTP_SEP_CHIPLET_ID_CPL_4 fields
Table 751. km_csr.OTP_SEP_CHIPLET_ID_CPL_5 fields
Table 752. km_csr.OTP_SEP_CHIPLET_ID_CPL_6 fields
Table 753. km_csr.OTP_SEP_CHIPLET_ID_CPL_7 fields
Table 754. km_csr.OTP_SEP_SIP_ID_VAL_0 fields

Table 755. km_csr.OTP_SEP_SIP_ID_VAL_1 fields
Table 756. km_csr.OTP_SEP_SIP_ID_VAL_2 fields
Table 757. km_csr.OTP_SEP_SIP_ID_VAL_3 fields
Table 758. km_csr.OTP_SEP_SIP_ID_VAL_4 fields
Table 759. km_csr.OTP_SEP_SIP_ID_VAL_5 fields
Table 760. km_csr.OTP_SEP_SIP_ID_VAL_6 fields
Table 761. km_csr.OTP_SEP_SIP_ID_VAL_7 fields
Table 762. km_csr.OTP_SEP_SIP_ID_CPL_0 fields
Table 763. km_csr.OTP_SEP_SIP_ID_CPL_1 fields
Table 764. km_csr.OTP_SEP_SIP_ID_CPL_2 fields
Table 765. km_csr.OTP_SEP_SIP_ID_CPL_3 fields
Table 766. km_csr.OTP_SEP_SIP_ID_CPL_4 fields
Table 767. km_csr.OTP_SEP_SIP_ID_CPL_5 fields
Table 768. km_csr.OTP_SEP_SIP_ID_CPL_6 fields
Table 769. km_csr.OTP_SEP_SIP_ID_CPL_7 fields
Table 770. km_csr.OTP_SEP_SYS_ID_VAL_0 fields
Table 771. km_csr.OTP_SEP_SYS_ID_VAL_1 fields
Table 772. km_csr.OTP_SEP_SYS_ID_VAL_2 fields
Table 773. km_csr.OTP_SEP_SYS_ID_VAL_3 fields
Table 774. km_csr.OTP_SEP_SYS_ID_VAL_4 fields
Table 775. km_csr.OTP_SEP_SYS_ID_VAL_5 fields
Table 776. km_csr.OTP_SEP_SYS_ID_VAL_6 fields
Table 777. km_csr.OTP_SEP_SYS_ID_VAL_7 fields
Table 778. km_csr.OTP_SEP_SYS_ID_CPL_0 fields
Table 779. km_csr.OTP_SEP_SYS_ID_CPL_1 fields
Table 780. km_csr.OTP_SEP_SYS_ID_CPL_2 fields
Table 781. km_csr.OTP_SEP_SYS_ID_CPL_3 fields
Table 782. km_csr.OTP_SEP_SYS_ID_CPL_4 fields
Table 783. km_csr.OTP_SEP_SYS_ID_CPL_5 fields
Table 784. km_csr.OTP_SEP_SYS_ID_CPL_6 fields
Table 785. km_csr.OTP_SEP_SYS_ID_CPL_7 fields
Table 786. Key and Policy Vault (KM Port) register list
Table 787. km_kpv.KEY_ENTRY[64].WORD[16] fields
Table 788. km_kpv.CTRL[64] fields
Table 789. km_kpv.KPV_SCRAMBLER_KEY fields
Table 790. km_kpv.KPV_SCRAMBLER_CTRL fields

Table 791. DRBG Sampler register list

Table 792. km_drbg_sampler.DATA fields

Table 793. km_drbg_sampler.CFG fields

Table 794. km_drbg_sampler.STATUS fields

Table 795. km_drbg_sampler.PREFETCH_DATA fields

Table 796. Key Manager Mailbox KM Side register list

Table 797. km_mailbox_km.KM_WRITE_DATA fields

Table 798. km_mailbox_km.KM_WRITE_SEPARATOR fields

Table 799. km_mailbox_km.KM_READ_DATA fields

Table 800. km_mailbox_km.KM_STATUS fields

Table 801. km_mailbox_km.KM_IRQ_STATUS fields

Table 802. km_mailbox_km.KM_IRQ_ENABLE fields

Table 803. km_mailbox_km.KM_CTRL fields

Table 804. Key Manager Mailbox SEP Side register list

Table 805. km_mailbox_sep.SEP_WRITE_DATA fields

Table 806. km_mailbox_sep.SEP_WRITE_SEPARATOR fields

Table 807. km_mailbox_sep.SEP_READ_DATA fields

Table 808. km_mailbox_sep.SEP_STATUS fields

Table 809. km_mailbox_sep.SEP_IRQ_STATUS fields

Table 810. km_mailbox_sep.SEP_IRQ_ENABLE fields

Table 811. km_mailbox_sep.SEP_CTRL fields

Table 812. Adams Bridge Key Sideload Registers register list

Table 813. abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] fields

Table 814. abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] fields

Table 815. abr_wrapper_key.MLDSA_SEED.KEY_CTRL fields

Table 816. abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] fields

Table 817. abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] fields

Table 818. abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL fields

Table 819. abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] fields

Table 820. abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] fields

Table 821. abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL fields

Table 822. abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] fields

Table 823. abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] fields

Table 824. abr_wrapper_key.MLKEM_MSG.KEY_CTRL fields

Table 825. abr_wrapper_key.KEY[8] fields

Table 826. abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL fields

Table 827. abr_wrapper_key.IRQ_STATUS fields
Table 828. abr_wrapper_key.IRQ_ENABLE fields
Table 829. AES Wrapper Key Registers register list
Table 830. aes_wrapper_key.KEY_SHARE0[8] fields
Table 831. aes_wrapper_key.KEY_SHARE1[8] fields
Table 832. aes_wrapper_key.KEY_CTRL fields
Table 833. HMAC Wrapper Key Registers register list
Table 834. hmac_wrapper_key.KEY_SHARE0[8] fields
Table 835. hmac_wrapper_key.KEY_SHARE1[8] fields
Table 836. hmac_wrapper_key.KEY_CTRL fields
Table 837. KMAC Wrapper Key Registers register list
Table 838. kmac_wrapper_key.KEY_SHARE0[8] fields
Table 839. kmac_wrapper_key.KEY_SHARE1[8] fields
Table 840. kmac_wrapper_key.KEY_CTRL fields
Table 841. OTBN Wrapper Key Registers register list
Table 842. otbn_wrapper_key.KEY_SHARE0[12] fields
Table 843. otbn_wrapper_key.KEY_SHARE1[12] fields
Table 844. otbn_wrapper_key.KEY_CTRL fields
Table 845. SEP CPU Logical Address Space
Table 846. SEP EL2 CPU Resources
Table 847. SEP Component Address Map
Table 848. SEP External register list
Table 849. sep_external.EFUSE_BANK_INIT_TIME fields
Table 850. SMC clock-domain responsibilities
Table 851. SMC reset sources and effects
Table 852. Reset synchronization domains
Table 853. Reset Unit Address Map register list
Table 854. reset_unit.SS_CONFIG fields
Table 855. reset_unit.SS_CONFIG_LOCK fields
Table 856. reset_unit.SS_COLD_RESET_N fields
Table 857. reset_unit.SS_WARM_RESET_N fields
Table 858. reset_unit.SS_CONFIG_HOLD fields
Table 859. reset_unit.SS_SRAM_HOLD fields
Table 860. reset_unit.SS_CRITICAL_HOLD fields
Table 861. reset_unit.SS_DEBUG_HOLD fields
Table 862. reset_unit.SS_RESET_COMPLETE fields

Table 863. reset_unit.SS_COLD_RESET_LOCK fields

Table 864. reset_unit.SS_FORCE_TO_REF_CLK fields

Table 865. reset_unit.SYNC_REG fields

Table 866. reset_unit.ISOLATE_REQ_REG fields

Table 867. reset_unit.ISOLATE_REQ_PINEN_REG fields

Table 868. reset_unit.ISOLATE_REQ_SMC_REG fields

Table 869. reset_unit.ISOLATE_REQ_SMCEN_REG fields

Table 870. reset_unit.ISOLATE_REQ_VIS fields

Table 871. reset_unit.ISOLATE_REQ_FLR_COUNTER_VALUE fields

Table 872. reset_unit.ISOLATE_REQ_FLR_RESET_COUNTER_VALUE fields

Table 873. SMC processor core features

Table 874. SMC cache and memory hierarchy

Table 875. SMC CPU system interfaces

Table 876. SMC Debug Module capability readbacks

Table 877. SMC Debug Module DMI register inventory

Table 878. CPU Control Address Map register list

Table 879. cpu_ctrl.RESET_VECTOR[4] fields

Table 880. cpu_ctrl.RESET_CTRL fields

Table 881. cpu_ctrl.CORE_RESET_PULSE_COUNT fields

Table 882. cpu_ctrl.RESET_TIMEOUT fields

Table 883. cpu_ctrl.REFERENCE_COUNTER fields

Table 884. cpu_ctrl.WDT_TIMEOUT fields

Table 885. cpu_ctrl.WDT_TIMEOUT_RESET fields

Table 886. cpu_ctrl.TEST_CTRL fields

Table 887. cpu_ctrl.SCRATCH[16] fields

Table 888. cpu_ctrl.WB_PC_CORE0[8] fields

Table 889. cpu_ctrl.WB_PC_CORE1[8] fields

Table 890. cpu_ctrl.WB_PC_CORE2[8] fields

Table 891. cpu_ctrl.WB_PC_CORE3[8] fields

Table 892. cpu_ctrl.SMC_ATTRIBUTES fields

Table 893. cpu_ctrl.MUTEX[4] fields

Table 894. cpu_ctrl.SEMA[4] fields

Table 895. cpu_ctrl.DUMMY_ROM_0 fields

Table 896. cpu_ctrl.DUMMY_ROM_1 fields

Table 897. cpu_ctrl.DUMMY_ROM_2 fields

Table 898. cpu_ctrl.DUMMY_ROM_3 fields

Table 899. cpu_ctrl.DUMMY_ROM_NULL[4] fields
Table 900. CLINT Address Map register list
Table 901. clint.MSIP[4] fields
Table 902. clint.MTIMECMP[4] fields
Table 903. clint.MTIME fields
Table 904. Debug Module Hart Interface register list
Table 905. debug_module.HALTED fields
Table 906. debug_module.GOING fields
Table 907. debug_module.RESUMING fields
Table 908. debug_module.EXCEPTION fields
Table 909. debug_module.WHERETO fields
Table 910. debug_module.ABSTRACT[5] fields
Table 911. debug_module.PROGBUF[16] fields
Table 912. debug_module.IMPEBREAK fields
Table 913. debug_module.DATA[4] fields
Table 914. debug_module.FLAGS fields
Table 915. debug_module.ROM[26] fields
Table 916. Bus Error Unit Address Map register list
Table 917. bus_error_unit.CAUSE fields
Table 918. bus_error_unit.PHYS_ADDR fields
Table 919. bus_error_unit.ENABLE fields
Table 920. bus_error_unit.PLIC_ENABLE fields
Table 921. bus_error_unit.ACCRUED_ENABLE fields
Table 922. bus_error_unit.LOCAL_ENABLE fields
Table 923. SMC Fabric Transaction Limits
Table 924. SMC Network Characteristics
Table 925. AXI Common Signal Widths
Table 926. AXI ID Widths by Fabric Stage
Table 927. SMC Fabric Traffic Managers
Table 928. SMC Fabric Traffic Subordinates
Table 929. SMC AXI Protection Bit Encoding
Table 930. SMC Source ID by Traffic Path
Table 931. AXI Alias Remap Address Map register list
Table 932. alias_remap.region_start fields
Table 933. alias_remap.region_end fields
Table 934. alias_remap.region_attrs fields

Table 935. Output Remap Address Map register list
Table 936. output_remap.region_attrs fields
Table 937. AXI Filter Control Address Map register list
Table 938. filter_ctrl.FILTER_CONFIG fields
Table 939. filter_ctrl.START_ADDR fields
Table 940. filter_ctrl.END_ADDR fields
Table 941. SMC interrupt-controller addresses
Table 942. PLIC input groups
Table 943. SMC indexed interrupt map
Table 944. PLIC Address Map register list
Table 945. plic.PRIORITY[337] fields
Table 946. plic.PENDING[11] fields
Table 947. plic.CORE0_MEIP_ENABLE[11] fields
Table 948. plic.CORE1_MEIP_ENABLE[11] fields
Table 949. plic.CORE2_MEIP_ENABLE[11] fields
Table 950. plic.CORE3_MEIP_ENABLE[11] fields
Table 951. plic.CORE0_SEIP_ENABLE[11] fields
Table 952. plic.CORE1_SEIP_ENABLE[11] fields
Table 953. plic.CORE2_SEIP_ENABLE[11] fields
Table 954. plic.CORE3_SEIP_ENABLE[11] fields
Table 955. plic.CORE0_MEIP_THRESHOLD fields
Table 956. plic.CORE0_MEIP_CLAIM_COMPLETE fields
Table 957. plic.CORE1_MEIP_THRESHOLD fields
Table 958. plic.CORE1_MEIP_CLAIM_COMPLETE fields
Table 959. plic.CORE2_MEIP_THRESHOLD fields
Table 960. plic.CORE2_MEIP_CLAIM_COMPLETE fields
Table 961. plic.CORE3_MEIP_THRESHOLD fields
Table 962. plic.CORE3_MEIP_CLAIM_COMPLETE fields
Table 963. plic.CORE0_SEIP_THRESHOLD fields
Table 964. plic.CORE0_SEIP_CLAIM_COMPLETE fields
Table 965. plic.CORE1_SEIP_THRESHOLD fields
Table 966. plic.CORE1_SEIP_CLAIM_COMPLETE fields
Table 967. plic.CORE2_SEIP_THRESHOLD fields
Table 968. plic.CORE2_SEIP_CLAIM_COMPLETE fields
Table 969. plic.CORE3_SEIP_THRESHOLD fields
Table 970. plic.CORE3_SEIP_CLAIM_COMPLETE fields

Table 971. SMC Peripheral Summary

Table 972. SMC DMA integration

Table 973. SMC DMA elaboration settings

Table 974. DMA register storage and effective field widths

Table 975. DMA status and notification

Table 976. iDMA Controller Address Map register list

Table 977. dma_ctrl.CONFIG fields

Table 978. dma_ctrl.STATUS_0 fields

Table 979. dma_ctrl.STATUS_1 fields

Table 980. dma_ctrl.STATUS_2 fields

Table 981. dma_ctrl.STATUS_3 fields

Table 982. dma_ctrl.STATUS_4 fields

Table 983. dma_ctrl.STATUS_5 fields

Table 984. dma_ctrl.STATUS_6 fields

Table 985. dma_ctrl.STATUS_7 fields

Table 986. dma_ctrl.STATUS_8 fields

Table 987. dma_ctrl.STATUS_9 fields

Table 988. dma_ctrl.STATUS_10 fields

Table 989. dma_ctrl.STATUS_11 fields

Table 990. dma_ctrl.STATUS_12 fields

Table 991. dma_ctrl.STATUS_13 fields

Table 992. dma_ctrl.STATUS_14 fields

Table 993. dma_ctrl.STATUS_15 fields

Table 994. dma_ctrl.NEXT_ID_0 fields

Table 995. dma_ctrl.NEXT_ID_1 fields

Table 996. dma_ctrl.NEXT_ID_2 fields

Table 997. dma_ctrl.NEXT_ID_3 fields

Table 998. dma_ctrl.NEXT_ID_4 fields

Table 999. dma_ctrl.NEXT_ID_5 fields

Table 1000. dma_ctrl.NEXT_ID_6 fields

Table 1001. dma_ctrl.NEXT_ID_7 fields

Table 1002. dma_ctrl.NEXT_ID_8 fields

Table 1003. dma_ctrl.NEXT_ID_9 fields

Table 1004. dma_ctrl.NEXT_ID_10 fields

Table 1005. dma_ctrl.NEXT_ID_11 fields

Table 1006. dma_ctrl.NEXT_ID_12 fields

Table 1007. dma_ctrl.NEXT_ID_13 fields

Table 1008. dma_ctrl.NEXT_ID_14 fields

Table 1009. dma_ctrl.NEXT_ID_15 fields

Table 1010. dma_ctrl.DONE_0 fields

Table 1011. dma_ctrl.DONE_1 fields

Table 1012. dma_ctrl.DONE_2 fields

Table 1013. dma_ctrl.DONE_3 fields

Table 1014. dma_ctrl.DONE_4 fields

Table 1015. dma_ctrl.DONE_5 fields

Table 1016. dma_ctrl.DONE_6 fields

Table 1017. dma_ctrl.DONE_7 fields

Table 1018. dma_ctrl.DONE_8 fields

Table 1019. dma_ctrl.DONE_9 fields

Table 1020. dma_ctrl.DONE_10 fields

Table 1021. dma_ctrl.DONE_11 fields

Table 1022. dma_ctrl.DONE_12 fields

Table 1023. dma_ctrl.DONE_13 fields

Table 1024. dma_ctrl.DONE_14 fields

Table 1025. dma_ctrl.DONE_15 fields

Table 1026. dma_ctrl.DST_ADDRESS_LO fields

Table 1027. dma_ctrl.DST_ADDRESS_HI fields

Table 1028. dma_ctrl.SRC_ADDRESS_LO fields

Table 1029. dma_ctrl.SRC_ADDRESS_HI fields

Table 1030. dma_ctrl.LENGTH_LO fields

Table 1031. dma_ctrl.LENGTH_HI fields

Table 1032. dma_ctrl.DST_STRIDE_LO fields

Table 1033. dma_ctrl.DST_STRIDE_HI fields

Table 1034. dma_ctrl.SRC_STRIDE_LO fields

Table 1035. dma_ctrl.SRC_STRIDE_HI fields

Table 1036. dma_ctrl.NUM_REPETITIONS_LO fields

Table 1037. dma_ctrl.NUM_REPETITIONS_HI fields

Table 1038. SMC Memory Zeroer Integration

Table 1039. SMC zeroer state machine

Table 1040. SMC zeroer AXI4 Master Characteristics

Table 1041. SMC zeroer Clock Gating Control

Table 1042. SMC zeroer Configuration and Control

Table 1043. SMC zeroer Data Generation and Control

Table 1044. Zeroer Control Address Map register list

Table 1045. zeroer_ctrl.DEST_ADDR fields

Table 1046. zeroer_ctrl.SIZE fields

Table 1047. zeroer_ctrl.CTRL_STATUS fields

Table 1048. Watchdog Timer Address Map register list

Table 1049. wdt.CTRL fields

Table 1050. wdt.COUNT fields

Table 1051. wdt.COUNT_HI fields

Table 1052. wdt.SCALED_COUNT fields

Table 1053. wdt.FEED fields

Table 1054. wdt.KEY fields

Table 1055. wdt.CMP fields

Table 1056. Mailbox interrupt sources

Table 1057. Mailbox access results

Table 1058. SMC Mailbox Aperture Layout

Table 1059. SEP Mailbox Aperture Layout

Table 1060. AXI-Lite Mailbox Address Map register list

Table 1061. axil_mailbox.WRITE_DATA fields

Table 1062. axil_mailbox.READ_DATA fields

Table 1063. axil_mailbox.STATUS fields

Table 1064. axil_mailbox.ERROR_FLAGS fields

Table 1065. axil_mailbox.WIRQT fields

Table 1066. axil_mailbox.RIRQT fields

Table 1067. axil_mailbox.IRQS fields

Table 1068. axil_mailbox.IRQEN fields

Table 1069. axil_mailbox.IRQP fields

Table 1070. axil_mailbox.CTRL fields

Table 1071. SMC AXI-Lite Mailbox Wrapper Address Map register list

Table 1072. axil_mailbox_smc_wrap.outbound_mailbox_0.WRITE_DATA fields

Table 1073. axil_mailbox_smc_wrap.outbound_mailbox_0.READ_DATA fields

Table 1074. axil_mailbox_smc_wrap.outbound_mailbox_0.STATUS fields

Table 1075. axil_mailbox_smc_wrap.outbound_mailbox_0.ERROR_FLAGS fields

Table 1076. axil_mailbox_smc_wrap.outbound_mailbox_0.WIRQT fields

Table 1077. axil_mailbox_smc_wrap.outbound_mailbox_0.RIRQT fields

Table 1078. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQS fields

Table 1079. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQEN fields
Table 1080. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQP fields
Table 1081. axil_mailbox_smc_wrap.outbound_mailbox_0.CTRL fields
Table 1082. axil_mailbox_smc_wrap.inbound_mailbox_0.WRITE_DATA fields
Table 1083. axil_mailbox_smc_wrap.inbound_mailbox_0.READ_DATA fields
Table 1084. axil_mailbox_smc_wrap.inbound_mailbox_0.STATUS fields
Table 1085. axil_mailbox_smc_wrap.inbound_mailbox_0.ERROR_FLAGS fields
Table 1086. axil_mailbox_smc_wrap.inbound_mailbox_0.WIRQT fields
Table 1087. axil_mailbox_smc_wrap.inbound_mailbox_0.RIRQT fields
Table 1088. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQS fields
Table 1089. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQEN fields
Table 1090. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQP fields
Table 1091. axil_mailbox_smc_wrap.inbound_mailbox_0.CTRL fields
Table 1092. axil_mailbox_smc_wrap.outbound_mailbox_1.WRITE_DATA fields
Table 1093. axil_mailbox_smc_wrap.outbound_mailbox_1.READ_DATA fields
Table 1094. axil_mailbox_smc_wrap.outbound_mailbox_1.STATUS fields
Table 1095. axil_mailbox_smc_wrap.outbound_mailbox_1.ERROR_FLAGS fields
Table 1096. axil_mailbox_smc_wrap.outbound_mailbox_1.WIRQT fields
Table 1097. axil_mailbox_smc_wrap.outbound_mailbox_1.RIRQT fields
Table 1098. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQS fields
Table 1099. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQEN fields
Table 1100. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQP fields
Table 1101. axil_mailbox_smc_wrap.outbound_mailbox_1.CTRL fields
Table 1102. axil_mailbox_smc_wrap.inbound_mailbox_1.WRITE_DATA fields
Table 1103. axil_mailbox_smc_wrap.inbound_mailbox_1.READ_DATA fields
Table 1104. axil_mailbox_smc_wrap.inbound_mailbox_1.STATUS fields
Table 1105. axil_mailbox_smc_wrap.inbound_mailbox_1.ERROR_FLAGS fields
Table 1106. axil_mailbox_smc_wrap.inbound_mailbox_1.WIRQT fields
Table 1107. axil_mailbox_smc_wrap.inbound_mailbox_1.RIRQT fields
Table 1108. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQS fields
Table 1109. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQEN fields
Table 1110. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQP fields
Table 1111. axil_mailbox_smc_wrap.inbound_mailbox_1.CTRL fields
Table 1112. axil_mailbox_smc_wrap.outbound_mailbox_2.WRITE_DATA fields
Table 1113. axil_mailbox_smc_wrap.outbound_mailbox_2.READ_DATA fields
Table 1114. axil_mailbox_smc_wrap.outbound_mailbox_2.STATUS fields

Table 1115. axil_mailbox_smc_wrap.outbound_mailbox_2.ERROR_FLAGS fields
Table 1116. axil_mailbox_smc_wrap.outbound_mailbox_2.WIRQT fields
Table 1117. axil_mailbox_smc_wrap.outbound_mailbox_2.RIRQT fields
Table 1118. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQS fields
Table 1119. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQEN fields
Table 1120. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQP fields
Table 1121. axil_mailbox_smc_wrap.outbound_mailbox_2.CTRL fields
Table 1122. axil_mailbox_smc_wrap.inbound_mailbox_2.WRITE_DATA fields
Table 1123. axil_mailbox_smc_wrap.inbound_mailbox_2.READ_DATA fields
Table 1124. axil_mailbox_smc_wrap.inbound_mailbox_2.STATUS fields
Table 1125. axil_mailbox_smc_wrap.inbound_mailbox_2.ERROR_FLAGS fields
Table 1126. axil_mailbox_smc_wrap.inbound_mailbox_2.WIRQT fields
Table 1127. axil_mailbox_smc_wrap.inbound_mailbox_2.RIRQT fields
Table 1128. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQS fields
Table 1129. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQEN fields
Table 1130. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQP fields
Table 1131. axil_mailbox_smc_wrap.inbound_mailbox_2.CTRL fields
Table 1132. axil_mailbox_smc_wrap.outbound_mailbox_3.WRITE_DATA fields
Table 1133. axil_mailbox_smc_wrap.outbound_mailbox_3.READ_DATA fields
Table 1134. axil_mailbox_smc_wrap.outbound_mailbox_3.STATUS fields
Table 1135. axil_mailbox_smc_wrap.outbound_mailbox_3.ERROR_FLAGS fields
Table 1136. axil_mailbox_smc_wrap.outbound_mailbox_3.WIRQT fields
Table 1137. axil_mailbox_smc_wrap.outbound_mailbox_3.RIRQT fields
Table 1138. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQS fields
Table 1139. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQEN fields
Table 1140. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQP fields
Table 1141. axil_mailbox_smc_wrap.outbound_mailbox_3.CTRL fields
Table 1142. axil_mailbox_smc_wrap.inbound_mailbox_3.WRITE_DATA fields
Table 1143. axil_mailbox_smc_wrap.inbound_mailbox_3.READ_DATA fields
Table 1144. axil_mailbox_smc_wrap.inbound_mailbox_3.STATUS fields
Table 1145. axil_mailbox_smc_wrap.inbound_mailbox_3.ERROR_FLAGS fields
Table 1146. axil_mailbox_smc_wrap.inbound_mailbox_3.WIRQT fields
Table 1147. axil_mailbox_smc_wrap.inbound_mailbox_3.RIRQT fields
Table 1148. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQS fields
Table 1149. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQEN fields
Table 1150. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQP fields

Table 1151. axil_mailbox_smc_wrap.inbound_mailbox_3.CTRL fields
Table 1152. axil_mailbox_smc_wrap.outbound_mailbox_4.WRITE_DATA fields
Table 1153. axil_mailbox_smc_wrap.outbound_mailbox_4.READ_DATA fields
Table 1154. axil_mailbox_smc_wrap.outbound_mailbox_4.STATUS fields
Table 1155. axil_mailbox_smc_wrap.outbound_mailbox_4.ERROR_FLAGS fields
Table 1156. axil_mailbox_smc_wrap.outbound_mailbox_4.WIRQT fields
Table 1157. axil_mailbox_smc_wrap.outbound_mailbox_4.RIRQT fields
Table 1158. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQS fields
Table 1159. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQEN fields
Table 1160. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQP fields
Table 1161. axil_mailbox_smc_wrap.outbound_mailbox_4.CTRL fields
Table 1162. axil_mailbox_smc_wrap.inbound_mailbox_4.WRITE_DATA fields
Table 1163. axil_mailbox_smc_wrap.inbound_mailbox_4.READ_DATA fields
Table 1164. axil_mailbox_smc_wrap.inbound_mailbox_4.STATUS fields
Table 1165. axil_mailbox_smc_wrap.inbound_mailbox_4.ERROR_FLAGS fields
Table 1166. axil_mailbox_smc_wrap.inbound_mailbox_4.WIRQT fields
Table 1167. axil_mailbox_smc_wrap.inbound_mailbox_4.RIRQT fields
Table 1168. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQS fields
Table 1169. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQEN fields
Table 1170. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQP fields
Table 1171. axil_mailbox_smc_wrap.inbound_mailbox_4.CTRL fields
Table 1172. axil_mailbox_smc_wrap.outbound_mailbox_5.WRITE_DATA fields
Table 1173. axil_mailbox_smc_wrap.outbound_mailbox_5.READ_DATA fields
Table 1174. axil_mailbox_smc_wrap.outbound_mailbox_5.STATUS fields
Table 1175. axil_mailbox_smc_wrap.outbound_mailbox_5.ERROR_FLAGS fields
Table 1176. axil_mailbox_smc_wrap.outbound_mailbox_5.WIRQT fields
Table 1177. axil_mailbox_smc_wrap.outbound_mailbox_5.RIRQT fields
Table 1178. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQS fields
Table 1179. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQEN fields
Table 1180. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQP fields
Table 1181. axil_mailbox_smc_wrap.outbound_mailbox_5.CTRL fields
Table 1182. axil_mailbox_smc_wrap.inbound_mailbox_5.WRITE_DATA fields
Table 1183. axil_mailbox_smc_wrap.inbound_mailbox_5.READ_DATA fields
Table 1184. axil_mailbox_smc_wrap.inbound_mailbox_5.STATUS fields
Table 1185. axil_mailbox_smc_wrap.inbound_mailbox_5.ERROR_FLAGS fields
Table 1186. axil_mailbox_smc_wrap.inbound_mailbox_5.WIRQT fields

Table 1187. axil_mailbox_smc_wrap.inbound_mailbox_5.RIRQT fields
Table 1188. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQS fields
Table 1189. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQEN fields
Table 1190. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQP fields
Table 1191. axil_mailbox_smc_wrap.inbound_mailbox_5.CTRL fields
Table 1192. axil_mailbox_smc_wrap.outbound_mailbox_6.WRITE_DATA fields
Table 1193. axil_mailbox_smc_wrap.outbound_mailbox_6.READ_DATA fields
Table 1194. axil_mailbox_smc_wrap.outbound_mailbox_6.STATUS fields
Table 1195. axil_mailbox_smc_wrap.outbound_mailbox_6.ERROR_FLAGS fields
Table 1196. axil_mailbox_smc_wrap.outbound_mailbox_6.WIRQT fields
Table 1197. axil_mailbox_smc_wrap.outbound_mailbox_6.RIRQT fields
Table 1198. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQS fields
Table 1199. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQEN fields
Table 1200. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQP fields
Table 1201. axil_mailbox_smc_wrap.outbound_mailbox_6.CTRL fields
Table 1202. axil_mailbox_smc_wrap.inbound_mailbox_6.WRITE_DATA fields
Table 1203. axil_mailbox_smc_wrap.inbound_mailbox_6.READ_DATA fields
Table 1204. axil_mailbox_smc_wrap.inbound_mailbox_6.STATUS fields
Table 1205. axil_mailbox_smc_wrap.inbound_mailbox_6.ERROR_FLAGS fields
Table 1206. axil_mailbox_smc_wrap.inbound_mailbox_6.WIRQT fields
Table 1207. axil_mailbox_smc_wrap.inbound_mailbox_6.RIRQT fields
Table 1208. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQS fields
Table 1209. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQEN fields
Table 1210. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQP fields
Table 1211. axil_mailbox_smc_wrap.inbound_mailbox_6.CTRL fields
Table 1212. axil_mailbox_smc_wrap.outbound_mailbox_7.WRITE_DATA fields
Table 1213. axil_mailbox_smc_wrap.outbound_mailbox_7.READ_DATA fields
Table 1214. axil_mailbox_smc_wrap.outbound_mailbox_7.STATUS fields
Table 1215. axil_mailbox_smc_wrap.outbound_mailbox_7.ERROR_FLAGS fields
Table 1216. axil_mailbox_smc_wrap.outbound_mailbox_7.WIRQT fields
Table 1217. axil_mailbox_smc_wrap.outbound_mailbox_7.RIRQT fields
Table 1218. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQS fields
Table 1219. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQEN fields
Table 1220. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQP fields
Table 1221. axil_mailbox_smc_wrap.outbound_mailbox_7.CTRL fields
Table 1222. axil_mailbox_smc_wrap.inbound_mailbox_7.WRITE_DATA fields

Table 1223. axil_mailbox_smc_wrap.inbound_mailbox_7.READ_DATA fields
Table 1224. axil_mailbox_smc_wrap.inbound_mailbox_7.STATUS fields
Table 1225. axil_mailbox_smc_wrap.inbound_mailbox_7.ERROR_FLAGS fields
Table 1226. axil_mailbox_smc_wrap.inbound_mailbox_7.WIRQT fields
Table 1227. axil_mailbox_smc_wrap.inbound_mailbox_7.RIRQT fields
Table 1228. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQS fields
Table 1229. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQEN fields
Table 1230. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQP fields
Table 1231. axil_mailbox_smc_wrap.inbound_mailbox_7.CTRL fields
Table 1232. axil_mailbox_smc_wrap.outbound_mailbox_8.WRITE_DATA fields
Table 1233. axil_mailbox_smc_wrap.outbound_mailbox_8.READ_DATA fields
Table 1234. axil_mailbox_smc_wrap.outbound_mailbox_8.STATUS fields
Table 1235. axil_mailbox_smc_wrap.outbound_mailbox_8.ERROR_FLAGS fields
Table 1236. axil_mailbox_smc_wrap.outbound_mailbox_8.WIRQT fields
Table 1237. axil_mailbox_smc_wrap.outbound_mailbox_8.RIRQT fields
Table 1238. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQS fields
Table 1239. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQEN fields
Table 1240. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQP fields
Table 1241. axil_mailbox_smc_wrap.outbound_mailbox_8.CTRL fields
Table 1242. axil_mailbox_smc_wrap.inbound_mailbox_8.WRITE_DATA fields
Table 1243. axil_mailbox_smc_wrap.inbound_mailbox_8.READ_DATA fields
Table 1244. axil_mailbox_smc_wrap.inbound_mailbox_8.STATUS fields
Table 1245. axil_mailbox_smc_wrap.inbound_mailbox_8.ERROR_FLAGS fields
Table 1246. axil_mailbox_smc_wrap.inbound_mailbox_8.WIRQT fields
Table 1247. axil_mailbox_smc_wrap.inbound_mailbox_8.RIRQT fields
Table 1248. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQS fields
Table 1249. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQEN fields
Table 1250. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQP fields
Table 1251. axil_mailbox_smc_wrap.inbound_mailbox_8.CTRL fields
Table 1252. axil_mailbox_smc_wrap.outbound_mailbox_9.WRITE_DATA fields
Table 1253. axil_mailbox_smc_wrap.outbound_mailbox_9.READ_DATA fields
Table 1254. axil_mailbox_smc_wrap.outbound_mailbox_9.STATUS fields
Table 1255. axil_mailbox_smc_wrap.outbound_mailbox_9.ERROR_FLAGS fields
Table 1256. axil_mailbox_smc_wrap.outbound_mailbox_9.WIRQT fields
Table 1257. axil_mailbox_smc_wrap.outbound_mailbox_9.RIRQT fields
Table 1258. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQS fields

Table 1259. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQEN fields
Table 1260. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQP fields
Table 1261. axil_mailbox_smc_wrap.outbound_mailbox_9.CTRL fields
Table 1262. axil_mailbox_smc_wrap.inbound_mailbox_9.WRITE_DATA fields
Table 1263. axil_mailbox_smc_wrap.inbound_mailbox_9.READ_DATA fields
Table 1264. axil_mailbox_smc_wrap.inbound_mailbox_9.STATUS fields
Table 1265. axil_mailbox_smc_wrap.inbound_mailbox_9.ERROR_FLAGS fields
Table 1266. axil_mailbox_smc_wrap.inbound_mailbox_9.WIRQT fields
Table 1267. axil_mailbox_smc_wrap.inbound_mailbox_9.RIRQT fields
Table 1268. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQS fields
Table 1269. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQEN fields
Table 1270. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQP fields
Table 1271. axil_mailbox_smc_wrap.inbound_mailbox_9.CTRL fields
Table 1272. axil_mailbox_smc_wrap.outbound_mailbox_10.WRITE_DATA fields
Table 1273. axil_mailbox_smc_wrap.outbound_mailbox_10.READ_DATA fields
Table 1274. axil_mailbox_smc_wrap.outbound_mailbox_10.STATUS fields
Table 1275. axil_mailbox_smc_wrap.outbound_mailbox_10.ERROR_FLAGS fields
Table 1276. axil_mailbox_smc_wrap.outbound_mailbox_10.WIRQT fields
Table 1277. axil_mailbox_smc_wrap.outbound_mailbox_10.RIRQT fields
Table 1278. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQS fields
Table 1279. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQEN fields
Table 1280. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQP fields
Table 1281. axil_mailbox_smc_wrap.outbound_mailbox_10.CTRL fields
Table 1282. axil_mailbox_smc_wrap.inbound_mailbox_10.WRITE_DATA fields
Table 1283. axil_mailbox_smc_wrap.inbound_mailbox_10.READ_DATA fields
Table 1284. axil_mailbox_smc_wrap.inbound_mailbox_10.STATUS fields
Table 1285. axil_mailbox_smc_wrap.inbound_mailbox_10.ERROR_FLAGS fields
Table 1286. axil_mailbox_smc_wrap.inbound_mailbox_10.WIRQT fields
Table 1287. axil_mailbox_smc_wrap.inbound_mailbox_10.RIRQT fields
Table 1288. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQS fields
Table 1289. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQEN fields
Table 1290. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQP fields
Table 1291. axil_mailbox_smc_wrap.inbound_mailbox_10.CTRL fields
Table 1292. axil_mailbox_smc_wrap.outbound_mailbox_11.WRITE_DATA fields
Table 1293. axil_mailbox_smc_wrap.outbound_mailbox_11.READ_DATA fields
Table 1294. axil_mailbox_smc_wrap.outbound_mailbox_11.STATUS fields

Table 1295. axil_mailbox_smc_wrap.outbound_mailbox_11.ERROR_FLAGS fields
Table 1296. axil_mailbox_smc_wrap.outbound_mailbox_11.WIRQT fields
Table 1297. axil_mailbox_smc_wrap.outbound_mailbox_11.RIRQT fields
Table 1298. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQS fields
Table 1299. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQEN fields
Table 1300. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQP fields
Table 1301. axil_mailbox_smc_wrap.outbound_mailbox_11.CTRL fields
Table 1302. axil_mailbox_smc_wrap.inbound_mailbox_11.WRITE_DATA fields
Table 1303. axil_mailbox_smc_wrap.inbound_mailbox_11.READ_DATA fields
Table 1304. axil_mailbox_smc_wrap.inbound_mailbox_11.STATUS fields
Table 1305. axil_mailbox_smc_wrap.inbound_mailbox_11.ERROR_FLAGS fields
Table 1306. axil_mailbox_smc_wrap.inbound_mailbox_11.WIRQT fields
Table 1307. axil_mailbox_smc_wrap.inbound_mailbox_11.RIRQT fields
Table 1308. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQS fields
Table 1309. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQEN fields
Table 1310. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQP fields
Table 1311. axil_mailbox_smc_wrap.inbound_mailbox_11.CTRL fields
Table 1312. axil_mailbox_smc_wrap.outbound_mailbox_12.WRITE_DATA fields
Table 1313. axil_mailbox_smc_wrap.outbound_mailbox_12.READ_DATA fields
Table 1314. axil_mailbox_smc_wrap.outbound_mailbox_12.STATUS fields
Table 1315. axil_mailbox_smc_wrap.outbound_mailbox_12.ERROR_FLAGS fields
Table 1316. axil_mailbox_smc_wrap.outbound_mailbox_12.WIRQT fields
Table 1317. axil_mailbox_smc_wrap.outbound_mailbox_12.RIRQT fields
Table 1318. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQS fields
Table 1319. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQEN fields
Table 1320. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQP fields
Table 1321. axil_mailbox_smc_wrap.outbound_mailbox_12.CTRL fields
Table 1322. axil_mailbox_smc_wrap.inbound_mailbox_12.WRITE_DATA fields
Table 1323. axil_mailbox_smc_wrap.inbound_mailbox_12.READ_DATA fields
Table 1324. axil_mailbox_smc_wrap.inbound_mailbox_12.STATUS fields
Table 1325. axil_mailbox_smc_wrap.inbound_mailbox_12.ERROR_FLAGS fields
Table 1326. axil_mailbox_smc_wrap.inbound_mailbox_12.WIRQT fields
Table 1327. axil_mailbox_smc_wrap.inbound_mailbox_12.RIRQT fields
Table 1328. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQS fields
Table 1329. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQEN fields
Table 1330. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQP fields

Table 1331. axil_mailbox_smc_wrap.inbound_mailbox_12.CTRL fields
Table 1332. axil_mailbox_smc_wrap.outbound_mailbox_13.WRITE_DATA fields
Table 1333. axil_mailbox_smc_wrap.outbound_mailbox_13.READ_DATA fields
Table 1334. axil_mailbox_smc_wrap.outbound_mailbox_13.STATUS fields
Table 1335. axil_mailbox_smc_wrap.outbound_mailbox_13.ERROR_FLAGS fields
Table 1336. axil_mailbox_smc_wrap.outbound_mailbox_13.WIRQT fields
Table 1337. axil_mailbox_smc_wrap.outbound_mailbox_13.RIRQT fields
Table 1338. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQS fields
Table 1339. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQEN fields
Table 1340. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQP fields
Table 1341. axil_mailbox_smc_wrap.outbound_mailbox_13.CTRL fields
Table 1342. axil_mailbox_smc_wrap.inbound_mailbox_13.WRITE_DATA fields
Table 1343. axil_mailbox_smc_wrap.inbound_mailbox_13.READ_DATA fields
Table 1344. axil_mailbox_smc_wrap.inbound_mailbox_13.STATUS fields
Table 1345. axil_mailbox_smc_wrap.inbound_mailbox_13.ERROR_FLAGS fields
Table 1346. axil_mailbox_smc_wrap.inbound_mailbox_13.WIRQT fields
Table 1347. axil_mailbox_smc_wrap.inbound_mailbox_13.RIRQT fields
Table 1348. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQS fields
Table 1349. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQEN fields
Table 1350. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQP fields
Table 1351. axil_mailbox_smc_wrap.inbound_mailbox_13.CTRL fields
Table 1352. axil_mailbox_smc_wrap.outbound_mailbox_14.WRITE_DATA fields
Table 1353. axil_mailbox_smc_wrap.outbound_mailbox_14.READ_DATA fields
Table 1354. axil_mailbox_smc_wrap.outbound_mailbox_14.STATUS fields
Table 1355. axil_mailbox_smc_wrap.outbound_mailbox_14.ERROR_FLAGS fields
Table 1356. axil_mailbox_smc_wrap.outbound_mailbox_14.WIRQT fields
Table 1357. axil_mailbox_smc_wrap.outbound_mailbox_14.RIRQT fields
Table 1358. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQS fields
Table 1359. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQEN fields
Table 1360. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQP fields
Table 1361. axil_mailbox_smc_wrap.outbound_mailbox_14.CTRL fields
Table 1362. axil_mailbox_smc_wrap.inbound_mailbox_14.WRITE_DATA fields
Table 1363. axil_mailbox_smc_wrap.inbound_mailbox_14.READ_DATA fields
Table 1364. axil_mailbox_smc_wrap.inbound_mailbox_14.STATUS fields
Table 1365. axil_mailbox_smc_wrap.inbound_mailbox_14.ERROR_FLAGS fields
Table 1366. axil_mailbox_smc_wrap.inbound_mailbox_14.WIRQT fields

Table 1367. axil_mailbox_smc_wrap.inbound_mailbox_14.RIRQT fields
Table 1368. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQS fields
Table 1369. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQEN fields
Table 1370. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQP fields
Table 1371. axil_mailbox_smc_wrap.inbound_mailbox_14.CTRL fields
Table 1372. axil_mailbox_smc_wrap.outbound_mailbox_15.WRITE_DATA fields
Table 1373. axil_mailbox_smc_wrap.outbound_mailbox_15.READ_DATA fields
Table 1374. axil_mailbox_smc_wrap.outbound_mailbox_15.STATUS fields
Table 1375. axil_mailbox_smc_wrap.outbound_mailbox_15.ERROR_FLAGS fields
Table 1376. axil_mailbox_smc_wrap.outbound_mailbox_15.WIRQT fields
Table 1377. axil_mailbox_smc_wrap.outbound_mailbox_15.RIRQT fields
Table 1378. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQS fields
Table 1379. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQEN fields
Table 1380. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQP fields
Table 1381. axil_mailbox_smc_wrap.outbound_mailbox_15.CTRL fields
Table 1382. axil_mailbox_smc_wrap.inbound_mailbox_15.WRITE_DATA fields
Table 1383. axil_mailbox_smc_wrap.inbound_mailbox_15.READ_DATA fields
Table 1384. axil_mailbox_smc_wrap.inbound_mailbox_15.STATUS fields
Table 1385. axil_mailbox_smc_wrap.inbound_mailbox_15.ERROR_FLAGS fields
Table 1386. axil_mailbox_smc_wrap.inbound_mailbox_15.WIRQT fields
Table 1387. axil_mailbox_smc_wrap.inbound_mailbox_15.RIRQT fields
Table 1388. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQS fields
Table 1389. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQEN fields
Table 1390. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQP fields
Table 1391. axil_mailbox_smc_wrap.inbound_mailbox_15.CTRL fields
Table 1392. axil_mailbox_smc_wrap.outbound_mailbox_16.WRITE_DATA fields
Table 1393. axil_mailbox_smc_wrap.outbound_mailbox_16.READ_DATA fields
Table 1394. axil_mailbox_smc_wrap.outbound_mailbox_16.STATUS fields
Table 1395. axil_mailbox_smc_wrap.outbound_mailbox_16.ERROR_FLAGS fields
Table 1396. axil_mailbox_smc_wrap.outbound_mailbox_16.WIRQT fields
Table 1397. axil_mailbox_smc_wrap.outbound_mailbox_16.RIRQT fields
Table 1398. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQS fields
Table 1399. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQEN fields
Table 1400. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQP fields
Table 1401. axil_mailbox_smc_wrap.outbound_mailbox_16.CTRL fields
Table 1402. axil_mailbox_smc_wrap.inbound_mailbox_16.WRITE_DATA fields

Table 1403. axil_mailbox_smc_wrap.inbound_mailbox_16.READ_DATA fields
Table 1404. axil_mailbox_smc_wrap.inbound_mailbox_16.STATUS fields
Table 1405. axil_mailbox_smc_wrap.inbound_mailbox_16.ERROR_FLAGS fields
Table 1406. axil_mailbox_smc_wrap.inbound_mailbox_16.WIRQT fields
Table 1407. axil_mailbox_smc_wrap.inbound_mailbox_16.RIRQT fields
Table 1408. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQS fields
Table 1409. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQEN fields
Table 1410. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQP fields
Table 1411. axil_mailbox_smc_wrap.inbound_mailbox_16.CTRL fields
Table 1412. axil_mailbox_smc_wrap.outbound_mailbox_17.WRITE_DATA fields
Table 1413. axil_mailbox_smc_wrap.outbound_mailbox_17.READ_DATA fields
Table 1414. axil_mailbox_smc_wrap.outbound_mailbox_17.STATUS fields
Table 1415. axil_mailbox_smc_wrap.outbound_mailbox_17.ERROR_FLAGS fields
Table 1416. axil_mailbox_smc_wrap.outbound_mailbox_17.WIRQT fields
Table 1417. axil_mailbox_smc_wrap.outbound_mailbox_17.RIRQT fields
Table 1418. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQS fields
Table 1419. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQEN fields
Table 1420. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQP fields
Table 1421. axil_mailbox_smc_wrap.outbound_mailbox_17.CTRL fields
Table 1422. axil_mailbox_smc_wrap.inbound_mailbox_17.WRITE_DATA fields
Table 1423. axil_mailbox_smc_wrap.inbound_mailbox_17.READ_DATA fields
Table 1424. axil_mailbox_smc_wrap.inbound_mailbox_17.STATUS fields
Table 1425. axil_mailbox_smc_wrap.inbound_mailbox_17.ERROR_FLAGS fields
Table 1426. axil_mailbox_smc_wrap.inbound_mailbox_17.WIRQT fields
Table 1427. axil_mailbox_smc_wrap.inbound_mailbox_17.RIRQT fields
Table 1428. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQS fields
Table 1429. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQEN fields
Table 1430. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQP fields
Table 1431. axil_mailbox_smc_wrap.inbound_mailbox_17.CTRL fields
Table 1432. axil_mailbox_smc_wrap.outbound_mailbox_18.WRITE_DATA fields
Table 1433. axil_mailbox_smc_wrap.outbound_mailbox_18.READ_DATA fields
Table 1434. axil_mailbox_smc_wrap.outbound_mailbox_18.STATUS fields
Table 1435. axil_mailbox_smc_wrap.outbound_mailbox_18.ERROR_FLAGS fields
Table 1436. axil_mailbox_smc_wrap.outbound_mailbox_18.WIRQT fields
Table 1437. axil_mailbox_smc_wrap.outbound_mailbox_18.RIRQT fields
Table 1438. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQS fields

Table 1439. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQEN fields
Table 1440. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQP fields
Table 1441. axil_mailbox_smc_wrap.outbound_mailbox_18.CTRL fields
Table 1442. axil_mailbox_smc_wrap.inbound_mailbox_18.WRITE_DATA fields
Table 1443. axil_mailbox_smc_wrap.inbound_mailbox_18.READ_DATA fields
Table 1444. axil_mailbox_smc_wrap.inbound_mailbox_18.STATUS fields
Table 1445. axil_mailbox_smc_wrap.inbound_mailbox_18.ERROR_FLAGS fields
Table 1446. axil_mailbox_smc_wrap.inbound_mailbox_18.WIRQT fields
Table 1447. axil_mailbox_smc_wrap.inbound_mailbox_18.RIRQT fields
Table 1448. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQS fields
Table 1449. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQEN fields
Table 1450. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQP fields
Table 1451. axil_mailbox_smc_wrap.inbound_mailbox_18.CTRL fields
Table 1452. axil_mailbox_smc_wrap.outbound_mailbox_19.WRITE_DATA fields
Table 1453. axil_mailbox_smc_wrap.outbound_mailbox_19.READ_DATA fields
Table 1454. axil_mailbox_smc_wrap.outbound_mailbox_19.STATUS fields
Table 1455. axil_mailbox_smc_wrap.outbound_mailbox_19.ERROR_FLAGS fields
Table 1456. axil_mailbox_smc_wrap.outbound_mailbox_19.WIRQT fields
Table 1457. axil_mailbox_smc_wrap.outbound_mailbox_19.RIRQT fields
Table 1458. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQS fields
Table 1459. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQEN fields
Table 1460. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQP fields
Table 1461. axil_mailbox_smc_wrap.outbound_mailbox_19.CTRL fields
Table 1462. axil_mailbox_smc_wrap.inbound_mailbox_19.WRITE_DATA fields
Table 1463. axil_mailbox_smc_wrap.inbound_mailbox_19.READ_DATA fields
Table 1464. axil_mailbox_smc_wrap.inbound_mailbox_19.STATUS fields
Table 1465. axil_mailbox_smc_wrap.inbound_mailbox_19.ERROR_FLAGS fields
Table 1466. axil_mailbox_smc_wrap.inbound_mailbox_19.WIRQT fields
Table 1467. axil_mailbox_smc_wrap.inbound_mailbox_19.RIRQT fields
Table 1468. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQS fields
Table 1469. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQEN fields
Table 1470. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQP fields
Table 1471. axil_mailbox_smc_wrap.inbound_mailbox_19.CTRL fields
Table 1472. axil_mailbox_smc_wrap.outbound_mailbox_20.WRITE_DATA fields
Table 1473. axil_mailbox_smc_wrap.outbound_mailbox_20.READ_DATA fields
Table 1474. axil_mailbox_smc_wrap.outbound_mailbox_20.STATUS fields

Table 1475. axil_mailbox_smc_wrap.outbound_mailbox_20.ERROR_FLAGS fields
Table 1476. axil_mailbox_smc_wrap.outbound_mailbox_20.WIRQT fields
Table 1477. axil_mailbox_smc_wrap.outbound_mailbox_20.RIRQT fields
Table 1478. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQS fields
Table 1479. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQEN fields
Table 1480. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQP fields
Table 1481. axil_mailbox_smc_wrap.outbound_mailbox_20.CTRL fields
Table 1482. axil_mailbox_smc_wrap.inbound_mailbox_20.WRITE_DATA fields
Table 1483. axil_mailbox_smc_wrap.inbound_mailbox_20.READ_DATA fields
Table 1484. axil_mailbox_smc_wrap.inbound_mailbox_20.STATUS fields
Table 1485. axil_mailbox_smc_wrap.inbound_mailbox_20.ERROR_FLAGS fields
Table 1486. axil_mailbox_smc_wrap.inbound_mailbox_20.WIRQT fields
Table 1487. axil_mailbox_smc_wrap.inbound_mailbox_20.RIRQT fields
Table 1488. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQS fields
Table 1489. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQEN fields
Table 1490. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQP fields
Table 1491. axil_mailbox_smc_wrap.inbound_mailbox_20.CTRL fields
Table 1492. axil_mailbox_smc_wrap.outbound_mailbox_21.WRITE_DATA fields
Table 1493. axil_mailbox_smc_wrap.outbound_mailbox_21.READ_DATA fields
Table 1494. axil_mailbox_smc_wrap.outbound_mailbox_21.STATUS fields
Table 1495. axil_mailbox_smc_wrap.outbound_mailbox_21.ERROR_FLAGS fields
Table 1496. axil_mailbox_smc_wrap.outbound_mailbox_21.WIRQT fields
Table 1497. axil_mailbox_smc_wrap.outbound_mailbox_21.RIRQT fields
Table 1498. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQS fields
Table 1499. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQEN fields
Table 1500. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQP fields
Table 1501. axil_mailbox_smc_wrap.outbound_mailbox_21.CTRL fields
Table 1502. axil_mailbox_smc_wrap.inbound_mailbox_21.WRITE_DATA fields
Table 1503. axil_mailbox_smc_wrap.inbound_mailbox_21.READ_DATA fields
Table 1504. axil_mailbox_smc_wrap.inbound_mailbox_21.STATUS fields
Table 1505. axil_mailbox_smc_wrap.inbound_mailbox_21.ERROR_FLAGS fields
Table 1506. axil_mailbox_smc_wrap.inbound_mailbox_21.WIRQT fields
Table 1507. axil_mailbox_smc_wrap.inbound_mailbox_21.RIRQT fields
Table 1508. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQS fields
Table 1509. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQEN fields
Table 1510. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQP fields

Table 1511. axil_mailbox_smc_wrap.inbound_mailbox_21.CTRL fields
Table 1512. axil_mailbox_smc_wrap.outbound_mailbox_22.WRITE_DATA fields
Table 1513. axil_mailbox_smc_wrap.outbound_mailbox_22.READ_DATA fields
Table 1514. axil_mailbox_smc_wrap.outbound_mailbox_22.STATUS fields
Table 1515. axil_mailbox_smc_wrap.outbound_mailbox_22.ERROR_FLAGS fields
Table 1516. axil_mailbox_smc_wrap.outbound_mailbox_22.WIRQT fields
Table 1517. axil_mailbox_smc_wrap.outbound_mailbox_22.RIRQT fields
Table 1518. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQS fields
Table 1519. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQEN fields
Table 1520. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQP fields
Table 1521. axil_mailbox_smc_wrap.outbound_mailbox_22.CTRL fields
Table 1522. axil_mailbox_smc_wrap.inbound_mailbox_22.WRITE_DATA fields
Table 1523. axil_mailbox_smc_wrap.inbound_mailbox_22.READ_DATA fields
Table 1524. axil_mailbox_smc_wrap.inbound_mailbox_22.STATUS fields
Table 1525. axil_mailbox_smc_wrap.inbound_mailbox_22.ERROR_FLAGS fields
Table 1526. axil_mailbox_smc_wrap.inbound_mailbox_22.WIRQT fields
Table 1527. axil_mailbox_smc_wrap.inbound_mailbox_22.RIRQT fields
Table 1528. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQS fields
Table 1529. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQEN fields
Table 1530. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQP fields
Table 1531. axil_mailbox_smc_wrap.inbound_mailbox_22.CTRL fields
Table 1532. axil_mailbox_smc_wrap.outbound_mailbox_23.WRITE_DATA fields
Table 1533. axil_mailbox_smc_wrap.outbound_mailbox_23.READ_DATA fields
Table 1534. axil_mailbox_smc_wrap.outbound_mailbox_23.STATUS fields
Table 1535. axil_mailbox_smc_wrap.outbound_mailbox_23.ERROR_FLAGS fields
Table 1536. axil_mailbox_smc_wrap.outbound_mailbox_23.WIRQT fields
Table 1537. axil_mailbox_smc_wrap.outbound_mailbox_23.RIRQT fields
Table 1538. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQS fields
Table 1539. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQEN fields
Table 1540. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQP fields
Table 1541. axil_mailbox_smc_wrap.outbound_mailbox_23.CTRL fields
Table 1542. axil_mailbox_smc_wrap.inbound_mailbox_23.WRITE_DATA fields
Table 1543. axil_mailbox_smc_wrap.inbound_mailbox_23.READ_DATA fields
Table 1544. axil_mailbox_smc_wrap.inbound_mailbox_23.STATUS fields
Table 1545. axil_mailbox_smc_wrap.inbound_mailbox_23.ERROR_FLAGS fields
Table 1546. axil_mailbox_smc_wrap.inbound_mailbox_23.WIRQT fields

Table 1547. axil_mailbox_smc_wrap.inbound_mailbox_23.RIRQT fields
Table 1548. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQS fields
Table 1549. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQEN fields
Table 1550. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQP fields
Table 1551. axil_mailbox_smc_wrap.inbound_mailbox_23.CTRL fields
Table 1552. axil_mailbox_smc_wrap.outbound_mailbox_24.WRITE_DATA fields
Table 1553. axil_mailbox_smc_wrap.outbound_mailbox_24.READ_DATA fields
Table 1554. axil_mailbox_smc_wrap.outbound_mailbox_24.STATUS fields
Table 1555. axil_mailbox_smc_wrap.outbound_mailbox_24.ERROR_FLAGS fields
Table 1556. axil_mailbox_smc_wrap.outbound_mailbox_24.WIRQT fields
Table 1557. axil_mailbox_smc_wrap.outbound_mailbox_24.RIRQT fields
Table 1558. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQS fields
Table 1559. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQEN fields
Table 1560. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQP fields
Table 1561. axil_mailbox_smc_wrap.outbound_mailbox_24.CTRL fields
Table 1562. axil_mailbox_smc_wrap.inbound_mailbox_24.WRITE_DATA fields
Table 1563. axil_mailbox_smc_wrap.inbound_mailbox_24.READ_DATA fields
Table 1564. axil_mailbox_smc_wrap.inbound_mailbox_24.STATUS fields
Table 1565. axil_mailbox_smc_wrap.inbound_mailbox_24.ERROR_FLAGS fields
Table 1566. axil_mailbox_smc_wrap.inbound_mailbox_24.WIRQT fields
Table 1567. axil_mailbox_smc_wrap.inbound_mailbox_24.RIRQT fields
Table 1568. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQS fields
Table 1569. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQEN fields
Table 1570. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQP fields
Table 1571. axil_mailbox_smc_wrap.inbound_mailbox_24.CTRL fields
Table 1572. axil_mailbox_smc_wrap.outbound_mailbox_25.WRITE_DATA fields
Table 1573. axil_mailbox_smc_wrap.outbound_mailbox_25.READ_DATA fields
Table 1574. axil_mailbox_smc_wrap.outbound_mailbox_25.STATUS fields
Table 1575. axil_mailbox_smc_wrap.outbound_mailbox_25.ERROR_FLAGS fields
Table 1576. axil_mailbox_smc_wrap.outbound_mailbox_25.WIRQT fields
Table 1577. axil_mailbox_smc_wrap.outbound_mailbox_25.RIRQT fields
Table 1578. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQS fields
Table 1579. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQEN fields
Table 1580. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQP fields
Table 1581. axil_mailbox_smc_wrap.outbound_mailbox_25.CTRL fields
Table 1582. axil_mailbox_smc_wrap.inbound_mailbox_25.WRITE_DATA fields

Table 1583. axil_mailbox_smc_wrap.inbound_mailbox_25.READ_DATA fields
Table 1584. axil_mailbox_smc_wrap.inbound_mailbox_25.STATUS fields
Table 1585. axil_mailbox_smc_wrap.inbound_mailbox_25.ERROR_FLAGS fields
Table 1586. axil_mailbox_smc_wrap.inbound_mailbox_25.WIRQT fields
Table 1587. axil_mailbox_smc_wrap.inbound_mailbox_25.RIRQT fields
Table 1588. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQS fields
Table 1589. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQEN fields
Table 1590. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQP fields
Table 1591. axil_mailbox_smc_wrap.inbound_mailbox_25.CTRL fields
Table 1592. axil_mailbox_smc_wrap.outbound_mailbox_26.WRITE_DATA fields
Table 1593. axil_mailbox_smc_wrap.outbound_mailbox_26.READ_DATA fields
Table 1594. axil_mailbox_smc_wrap.outbound_mailbox_26.STATUS fields
Table 1595. axil_mailbox_smc_wrap.outbound_mailbox_26.ERROR_FLAGS fields
Table 1596. axil_mailbox_smc_wrap.outbound_mailbox_26.WIRQT fields
Table 1597. axil_mailbox_smc_wrap.outbound_mailbox_26.RIRQT fields
Table 1598. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQS fields
Table 1599. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQEN fields
Table 1600. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQP fields
Table 1601. axil_mailbox_smc_wrap.outbound_mailbox_26.CTRL fields
Table 1602. axil_mailbox_smc_wrap.inbound_mailbox_26.WRITE_DATA fields
Table 1603. axil_mailbox_smc_wrap.inbound_mailbox_26.READ_DATA fields
Table 1604. axil_mailbox_smc_wrap.inbound_mailbox_26.STATUS fields
Table 1605. axil_mailbox_smc_wrap.inbound_mailbox_26.ERROR_FLAGS fields
Table 1606. axil_mailbox_smc_wrap.inbound_mailbox_26.WIRQT fields
Table 1607. axil_mailbox_smc_wrap.inbound_mailbox_26.RIRQT fields
Table 1608. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQS fields
Table 1609. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQEN fields
Table 1610. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQP fields
Table 1611. axil_mailbox_smc_wrap.inbound_mailbox_26.CTRL fields
Table 1612. axil_mailbox_smc_wrap.outbound_mailbox_27.WRITE_DATA fields
Table 1613. axil_mailbox_smc_wrap.outbound_mailbox_27.READ_DATA fields
Table 1614. axil_mailbox_smc_wrap.outbound_mailbox_27.STATUS fields
Table 1615. axil_mailbox_smc_wrap.outbound_mailbox_27.ERROR_FLAGS fields
Table 1616. axil_mailbox_smc_wrap.outbound_mailbox_27.WIRQT fields
Table 1617. axil_mailbox_smc_wrap.outbound_mailbox_27.RIRQT fields
Table 1618. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQS fields

Table 1619. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQEN fields
Table 1620. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQP fields
Table 1621. axil_mailbox_smc_wrap.outbound_mailbox_27.CTRL fields
Table 1622. axil_mailbox_smc_wrap.inbound_mailbox_27.WRITE_DATA fields
Table 1623. axil_mailbox_smc_wrap.inbound_mailbox_27.READ_DATA fields
Table 1624. axil_mailbox_smc_wrap.inbound_mailbox_27.STATUS fields
Table 1625. axil_mailbox_smc_wrap.inbound_mailbox_27.ERROR_FLAGS fields
Table 1626. axil_mailbox_smc_wrap.inbound_mailbox_27.WIRQT fields
Table 1627. axil_mailbox_smc_wrap.inbound_mailbox_27.RIRQT fields
Table 1628. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQS fields
Table 1629. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQEN fields
Table 1630. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQP fields
Table 1631. axil_mailbox_smc_wrap.inbound_mailbox_27.CTRL fields
Table 1632. axil_mailbox_smc_wrap.outbound_mailbox_28.WRITE_DATA fields
Table 1633. axil_mailbox_smc_wrap.outbound_mailbox_28.READ_DATA fields
Table 1634. axil_mailbox_smc_wrap.outbound_mailbox_28.STATUS fields
Table 1635. axil_mailbox_smc_wrap.outbound_mailbox_28.ERROR_FLAGS fields
Table 1636. axil_mailbox_smc_wrap.outbound_mailbox_28.WIRQT fields
Table 1637. axil_mailbox_smc_wrap.outbound_mailbox_28.RIRQT fields
Table 1638. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQS fields
Table 1639. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQEN fields
Table 1640. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQP fields
Table 1641. axil_mailbox_smc_wrap.outbound_mailbox_28.CTRL fields
Table 1642. axil_mailbox_smc_wrap.inbound_mailbox_28.WRITE_DATA fields
Table 1643. axil_mailbox_smc_wrap.inbound_mailbox_28.READ_DATA fields
Table 1644. axil_mailbox_smc_wrap.inbound_mailbox_28.STATUS fields
Table 1645. axil_mailbox_smc_wrap.inbound_mailbox_28.ERROR_FLAGS fields
Table 1646. axil_mailbox_smc_wrap.inbound_mailbox_28.WIRQT fields
Table 1647. axil_mailbox_smc_wrap.inbound_mailbox_28.RIRQT fields
Table 1648. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQS fields
Table 1649. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQEN fields
Table 1650. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQP fields
Table 1651. axil_mailbox_smc_wrap.inbound_mailbox_28.CTRL fields
Table 1652. axil_mailbox_smc_wrap.outbound_mailbox_29.WRITE_DATA fields
Table 1653. axil_mailbox_smc_wrap.outbound_mailbox_29.READ_DATA fields
Table 1654. axil_mailbox_smc_wrap.outbound_mailbox_29.STATUS fields

Table 1655. axil_mailbox_smc_wrap.outbound_mailbox_29.ERROR_FLAGS fields
Table 1656. axil_mailbox_smc_wrap.outbound_mailbox_29.WIRQT fields
Table 1657. axil_mailbox_smc_wrap.outbound_mailbox_29.RIRQT fields
Table 1658. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQS fields
Table 1659. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQEN fields
Table 1660. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQP fields
Table 1661. axil_mailbox_smc_wrap.outbound_mailbox_29.CTRL fields
Table 1662. axil_mailbox_smc_wrap.inbound_mailbox_29.WRITE_DATA fields
Table 1663. axil_mailbox_smc_wrap.inbound_mailbox_29.READ_DATA fields
Table 1664. axil_mailbox_smc_wrap.inbound_mailbox_29.STATUS fields
Table 1665. axil_mailbox_smc_wrap.inbound_mailbox_29.ERROR_FLAGS fields
Table 1666. axil_mailbox_smc_wrap.inbound_mailbox_29.WIRQT fields
Table 1667. axil_mailbox_smc_wrap.inbound_mailbox_29.RIRQT fields
Table 1668. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQS fields
Table 1669. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQEN fields
Table 1670. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQP fields
Table 1671. axil_mailbox_smc_wrap.inbound_mailbox_29.CTRL fields
Table 1672. axil_mailbox_smc_wrap.outbound_mailbox_30.WRITE_DATA fields
Table 1673. axil_mailbox_smc_wrap.outbound_mailbox_30.READ_DATA fields
Table 1674. axil_mailbox_smc_wrap.outbound_mailbox_30.STATUS fields
Table 1675. axil_mailbox_smc_wrap.outbound_mailbox_30.ERROR_FLAGS fields
Table 1676. axil_mailbox_smc_wrap.outbound_mailbox_30.WIRQT fields
Table 1677. axil_mailbox_smc_wrap.outbound_mailbox_30.RIRQT fields
Table 1678. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQS fields
Table 1679. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQEN fields
Table 1680. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQP fields
Table 1681. axil_mailbox_smc_wrap.outbound_mailbox_30.CTRL fields
Table 1682. axil_mailbox_smc_wrap.inbound_mailbox_30.WRITE_DATA fields
Table 1683. axil_mailbox_smc_wrap.inbound_mailbox_30.READ_DATA fields
Table 1684. axil_mailbox_smc_wrap.inbound_mailbox_30.STATUS fields
Table 1685. axil_mailbox_smc_wrap.inbound_mailbox_30.ERROR_FLAGS fields
Table 1686. axil_mailbox_smc_wrap.inbound_mailbox_30.WIRQT fields
Table 1687. axil_mailbox_smc_wrap.inbound_mailbox_30.RIRQT fields
Table 1688. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQS fields
Table 1689. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQEN fields
Table 1690. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQP fields

Table 1691. axil_mailbox_smc_wrap.inbound_mailbox_30.CTRL fields
Table 1692. axil_mailbox_smc_wrap.outbound_mailbox_31.WRITE_DATA fields
Table 1693. axil_mailbox_smc_wrap.outbound_mailbox_31.READ_DATA fields
Table 1694. axil_mailbox_smc_wrap.outbound_mailbox_31.STATUS fields
Table 1695. axil_mailbox_smc_wrap.outbound_mailbox_31.ERROR_FLAGS fields
Table 1696. axil_mailbox_smc_wrap.outbound_mailbox_31.WIRQT fields
Table 1697. axil_mailbox_smc_wrap.outbound_mailbox_31.RIRQT fields
Table 1698. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQS fields
Table 1699. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQEN fields
Table 1700. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQP fields
Table 1701. axil_mailbox_smc_wrap.outbound_mailbox_31.CTRL fields
Table 1702. axil_mailbox_smc_wrap.inbound_mailbox_31.WRITE_DATA fields
Table 1703. axil_mailbox_smc_wrap.inbound_mailbox_31.READ_DATA fields
Table 1704. axil_mailbox_smc_wrap.inbound_mailbox_31.STATUS fields
Table 1705. axil_mailbox_smc_wrap.inbound_mailbox_31.ERROR_FLAGS fields
Table 1706. axil_mailbox_smc_wrap.inbound_mailbox_31.WIRQT fields
Table 1707. axil_mailbox_smc_wrap.inbound_mailbox_31.RIRQT fields
Table 1708. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQS fields
Table 1709. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQEN fields
Table 1710. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQP fields
Table 1711. axil_mailbox_smc_wrap.inbound_mailbox_31.CTRL fields
Table 1712. SEP AXI-Lite Mailbox Wrapper Address Map register list
Table 1713. axil_mailbox_sep_wrap.outbound_mailbox_0.WRITE_DATA fields
Table 1714. axil_mailbox_sep_wrap.outbound_mailbox_0.READ_DATA fields
Table 1715. axil_mailbox_sep_wrap.outbound_mailbox_0.STATUS fields
Table 1716. axil_mailbox_sep_wrap.outbound_mailbox_0.ERROR_FLAGS fields
Table 1717. axil_mailbox_sep_wrap.outbound_mailbox_0.WIRQT fields
Table 1718. axil_mailbox_sep_wrap.outbound_mailbox_0.RIRQT fields
Table 1719. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQS fields
Table 1720. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQEN fields
Table 1721. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQP fields
Table 1722. axil_mailbox_sep_wrap.outbound_mailbox_0.CTRL fields
Table 1723. axil_mailbox_sep_wrap.inbound_mailbox_0.WRITE_DATA fields
Table 1724. axil_mailbox_sep_wrap.inbound_mailbox_0.READ_DATA fields
Table 1725. axil_mailbox_sep_wrap.inbound_mailbox_0.STATUS fields
Table 1726. axil_mailbox_sep_wrap.inbound_mailbox_0.ERROR_FLAGS fields

Table 1727. axil_mailbox_sep_wrap.inbound_mailbox_0.WIRQT fields
Table 1728. axil_mailbox_sep_wrap.inbound_mailbox_0.RIRQT fields
Table 1729. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQS fields
Table 1730. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQEN fields
Table 1731. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQP fields
Table 1732. axil_mailbox_sep_wrap.inbound_mailbox_0.CTRL fields
Table 1733. axil_mailbox_sep_wrap.outbound_mailbox_1.WRITE_DATA fields
Table 1734. axil_mailbox_sep_wrap.outbound_mailbox_1.READ_DATA fields
Table 1735. axil_mailbox_sep_wrap.outbound_mailbox_1.STATUS fields
Table 1736. axil_mailbox_sep_wrap.outbound_mailbox_1.ERROR_FLAGS fields
Table 1737. axil_mailbox_sep_wrap.outbound_mailbox_1.WIRQT fields
Table 1738. axil_mailbox_sep_wrap.outbound_mailbox_1.RIRQT fields
Table 1739. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQS fields
Table 1740. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQEN fields
Table 1741. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQP fields
Table 1742. axil_mailbox_sep_wrap.outbound_mailbox_1.CTRL fields
Table 1743. axil_mailbox_sep_wrap.inbound_mailbox_1.WRITE_DATA fields
Table 1744. axil_mailbox_sep_wrap.inbound_mailbox_1.READ_DATA fields
Table 1745. axil_mailbox_sep_wrap.inbound_mailbox_1.STATUS fields
Table 1746. axil_mailbox_sep_wrap.inbound_mailbox_1.ERROR_FLAGS fields
Table 1747. axil_mailbox_sep_wrap.inbound_mailbox_1.WIRQT fields
Table 1748. axil_mailbox_sep_wrap.inbound_mailbox_1.RIRQT fields
Table 1749. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQS fields
Table 1750. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQEN fields
Table 1751. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQP fields
Table 1752. axil_mailbox_sep_wrap.inbound_mailbox_1.CTRL fields
Table 1753. axil_mailbox_sep_wrap.outbound_mailbox_2.WRITE_DATA fields
Table 1754. axil_mailbox_sep_wrap.outbound_mailbox_2.READ_DATA fields
Table 1755. axil_mailbox_sep_wrap.outbound_mailbox_2.STATUS fields
Table 1756. axil_mailbox_sep_wrap.outbound_mailbox_2.ERROR_FLAGS fields
Table 1757. axil_mailbox_sep_wrap.outbound_mailbox_2.WIRQT fields
Table 1758. axil_mailbox_sep_wrap.outbound_mailbox_2.RIRQT fields
Table 1759. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQS fields
Table 1760. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQEN fields
Table 1761. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQP fields
Table 1762. axil_mailbox_sep_wrap.outbound_mailbox_2.CTRL fields

Table 1763. axil_mailbox_sep_wrap.inbound_mailbox_2.WRITE_DATA fields
Table 1764. axil_mailbox_sep_wrap.inbound_mailbox_2.READ_DATA fields
Table 1765. axil_mailbox_sep_wrap.inbound_mailbox_2.STATUS fields
Table 1766. axil_mailbox_sep_wrap.inbound_mailbox_2.ERROR_FLAGS fields
Table 1767. axil_mailbox_sep_wrap.inbound_mailbox_2.WIRQT fields
Table 1768. axil_mailbox_sep_wrap.inbound_mailbox_2.RIRQT fields
Table 1769. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQS fields
Table 1770. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQEN fields
Table 1771. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQP fields
Table 1772. axil_mailbox_sep_wrap.inbound_mailbox_2.CTRL fields
Table 1773. axil_mailbox_sep_wrap.outbound_mailbox_3.WRITE_DATA fields
Table 1774. axil_mailbox_sep_wrap.outbound_mailbox_3.READ_DATA fields
Table 1775. axil_mailbox_sep_wrap.outbound_mailbox_3.STATUS fields
Table 1776. axil_mailbox_sep_wrap.outbound_mailbox_3.ERROR_FLAGS fields
Table 1777. axil_mailbox_sep_wrap.outbound_mailbox_3.WIRQT fields
Table 1778. axil_mailbox_sep_wrap.outbound_mailbox_3.RIRQT fields
Table 1779. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQS fields
Table 1780. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQEN fields
Table 1781. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQP fields
Table 1782. axil_mailbox_sep_wrap.outbound_mailbox_3.CTRL fields
Table 1783. axil_mailbox_sep_wrap.inbound_mailbox_3.WRITE_DATA fields
Table 1784. axil_mailbox_sep_wrap.inbound_mailbox_3.READ_DATA fields
Table 1785. axil_mailbox_sep_wrap.inbound_mailbox_3.STATUS fields
Table 1786. axil_mailbox_sep_wrap.inbound_mailbox_3.ERROR_FLAGS fields
Table 1787. axil_mailbox_sep_wrap.inbound_mailbox_3.WIRQT fields
Table 1788. axil_mailbox_sep_wrap.inbound_mailbox_3.RIRQT fields
Table 1789. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQS fields
Table 1790. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQEN fields
Table 1791. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQP fields
Table 1792. axil_mailbox_sep_wrap.inbound_mailbox_3.CTRL fields
Table 1793. axil_mailbox_sep_wrap.outbound_mailbox_4.WRITE_DATA fields
Table 1794. axil_mailbox_sep_wrap.outbound_mailbox_4.READ_DATA fields
Table 1795. axil_mailbox_sep_wrap.outbound_mailbox_4.STATUS fields
Table 1796. axil_mailbox_sep_wrap.outbound_mailbox_4.ERROR_FLAGS fields
Table 1797. axil_mailbox_sep_wrap.outbound_mailbox_4.WIRQT fields
Table 1798. axil_mailbox_sep_wrap.outbound_mailbox_4.RIRQT fields

Table 1799. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQS fields
Table 1800. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQEN fields
Table 1801. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQP fields
Table 1802. axil_mailbox_sep_wrap.outbound_mailbox_4.CTRL fields
Table 1803. axil_mailbox_sep_wrap.inbound_mailbox_4.WRITE_DATA fields
Table 1804. axil_mailbox_sep_wrap.inbound_mailbox_4.READ_DATA fields
Table 1805. axil_mailbox_sep_wrap.inbound_mailbox_4.STATUS fields
Table 1806. axil_mailbox_sep_wrap.inbound_mailbox_4.ERROR_FLAGS fields
Table 1807. axil_mailbox_sep_wrap.inbound_mailbox_4.WIRQT fields
Table 1808. axil_mailbox_sep_wrap.inbound_mailbox_4.RIRQT fields
Table 1809. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQS fields
Table 1810. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQEN fields
Table 1811. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQP fields
Table 1812. axil_mailbox_sep_wrap.inbound_mailbox_4.CTRL fields
Table 1813. axil_mailbox_sep_wrap.outbound_mailbox_5.WRITE_DATA fields
Table 1814. axil_mailbox_sep_wrap.outbound_mailbox_5.READ_DATA fields
Table 1815. axil_mailbox_sep_wrap.outbound_mailbox_5.STATUS fields
Table 1816. axil_mailbox_sep_wrap.outbound_mailbox_5.ERROR_FLAGS fields
Table 1817. axil_mailbox_sep_wrap.outbound_mailbox_5.WIRQT fields
Table 1818. axil_mailbox_sep_wrap.outbound_mailbox_5.RIRQT fields
Table 1819. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQS fields
Table 1820. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQEN fields
Table 1821. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQP fields
Table 1822. axil_mailbox_sep_wrap.outbound_mailbox_5.CTRL fields
Table 1823. axil_mailbox_sep_wrap.inbound_mailbox_5.WRITE_DATA fields
Table 1824. axil_mailbox_sep_wrap.inbound_mailbox_5.READ_DATA fields
Table 1825. axil_mailbox_sep_wrap.inbound_mailbox_5.STATUS fields
Table 1826. axil_mailbox_sep_wrap.inbound_mailbox_5.ERROR_FLAGS fields
Table 1827. axil_mailbox_sep_wrap.inbound_mailbox_5.WIRQT fields
Table 1828. axil_mailbox_sep_wrap.inbound_mailbox_5.RIRQT fields
Table 1829. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQS fields
Table 1830. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQEN fields
Table 1831. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQP fields
Table 1832. axil_mailbox_sep_wrap.inbound_mailbox_5.CTRL fields
Table 1833. axil_mailbox_sep_wrap.outbound_mailbox_6.WRITE_DATA fields
Table 1834. axil_mailbox_sep_wrap.outbound_mailbox_6.READ_DATA fields

Table 1835. axil_mailbox_sep_wrap.outbound_mailbox_6.STATUS fields
Table 1836. axil_mailbox_sep_wrap.outbound_mailbox_6.ERROR_FLAGS fields
Table 1837. axil_mailbox_sep_wrap.outbound_mailbox_6.WIRQT fields
Table 1838. axil_mailbox_sep_wrap.outbound_mailbox_6.RIRQT fields
Table 1839. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQS fields
Table 1840. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQEN fields
Table 1841. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQP fields
Table 1842. axil_mailbox_sep_wrap.outbound_mailbox_6.CTRL fields
Table 1843. axil_mailbox_sep_wrap.inbound_mailbox_6.WRITE_DATA fields
Table 1844. axil_mailbox_sep_wrap.inbound_mailbox_6.READ_DATA fields
Table 1845. axil_mailbox_sep_wrap.inbound_mailbox_6.STATUS fields
Table 1846. axil_mailbox_sep_wrap.inbound_mailbox_6.ERROR_FLAGS fields
Table 1847. axil_mailbox_sep_wrap.inbound_mailbox_6.WIRQT fields
Table 1848. axil_mailbox_sep_wrap.inbound_mailbox_6.RIRQT fields
Table 1849. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQS fields
Table 1850. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQEN fields
Table 1851. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQP fields
Table 1852. axil_mailbox_sep_wrap.inbound_mailbox_6.CTRL fields
Table 1853. axil_mailbox_sep_wrap.outbound_mailbox_7.WRITE_DATA fields
Table 1854. axil_mailbox_sep_wrap.outbound_mailbox_7.READ_DATA fields
Table 1855. axil_mailbox_sep_wrap.outbound_mailbox_7.STATUS fields
Table 1856. axil_mailbox_sep_wrap.outbound_mailbox_7.ERROR_FLAGS fields
Table 1857. axil_mailbox_sep_wrap.outbound_mailbox_7.WIRQT fields
Table 1858. axil_mailbox_sep_wrap.outbound_mailbox_7.RIRQT fields
Table 1859. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQS fields
Table 1860. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQEN fields
Table 1861. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQP fields
Table 1862. axil_mailbox_sep_wrap.outbound_mailbox_7.CTRL fields
Table 1863. axil_mailbox_sep_wrap.inbound_mailbox_7.WRITE_DATA fields
Table 1864. axil_mailbox_sep_wrap.inbound_mailbox_7.READ_DATA fields
Table 1865. axil_mailbox_sep_wrap.inbound_mailbox_7.STATUS fields
Table 1866. axil_mailbox_sep_wrap.inbound_mailbox_7.ERROR_FLAGS fields
Table 1867. axil_mailbox_sep_wrap.inbound_mailbox_7.WIRQT fields
Table 1868. axil_mailbox_sep_wrap.inbound_mailbox_7.RIRQT fields
Table 1869. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQS fields
Table 1870. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQEN fields

Table 1871. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQP fields

Table 1872. axil_mailbox_sep_wrap.inbound_mailbox_7.CTRL fields

Table 1873. OCTS credit management

Table 1874. OCTS pulse states

Table 1875. OCTS register offsets and reset values

Table 1876. OCTS System Timer Address Map register list

Table 1877. system_timer_octs.TIMER_START fields

Table 1878. system_timer_octs.CTRL fields

Table 1879. system_timer_octs.STATUS fields

Table 1880. system_timer_octs.TIMER_PRESET_LO fields

Table 1881. system_timer_octs.TIMER_PRESET_HI fields

Table 1882. system_timer_octs.TIMER_COUNT_LO fields

Table 1883. system_timer_octs.TIMER_COUNT_HI fields

Table 1884. system_timer_octs.CREDIT_EXPIRED fields

Table 1885. system_timer_octs.TIMER_GPIO_ENABLE fields

Table 1886. eFuse destination endpoints

Table 1887. eFuse shadow-register features

Table 1888. eFuse command type macros

Table 1889. Example Hardware Interfaces by OTP Variant

Table 1890. eFuse lock-field encoding

Table 1891. eFuse hardware and software locks

Table 1892. eFuse security-disable behavior

Table 1893. eFuse lifecycle encodings

Table 1894. eFuse JTAG permissions (SMC)

Table 1895. eFuse secure-test behavior

Table 1896. SEP eFuse Placement

Table 1897. SMC eFuse Placement

Table 1898. Key Manager eFuse Pass-Through

Table 1899. Efuse address map of physical efuse macro register list

Table 1900. smc_efuse_map.LOCKS fields

Table 1901. smc_efuse_map.JTAG_PUBLIC_IDENTITY fields

Table 1902. smc_efuse_map.I2C_I3C_ID[9] fields

Table 1903. smc_efuse_map.SMC_CONFIG fields

Table 1904. smc_efuse_map.OCCP_TRANSPORT_TIMEOUT fields

Table 1905. smc_efuse_map.SPARE[28] fields

Table 1906. eFuse Token Registers Address Map register list

Table 1907. efuse_mmr.RMA_SIP_TOKEN_I[8] fields

Table 1908. efuse_mmr.RMA_CHIPLET_TOKEN_I[8] fields

Table 1909. efuse_mmr.SEC_DISABLE_TOKEN_I[8] fields

Table 1910. efuse_mmr.TOKEN_EOP fields

Table 1911. efuse_mmr.RMA_SIP_TOKEN_MATCH fields

Table 1912. efuse_mmr.RMA_CHIPLET_TOKEN_MATCH fields

Table 1913. efuse_mmr.SEC_DISABLE_TOKEN_MATCH fields

Table 1914. efuse_mmr.TOKEN_MATCH_FAULT fields

Table 1915. eFuse Interface Control Address Map register list

Table 1916. efuse_interface_ctrl.EFUSE_INTERFACE_CTRL_STATUS fields

Table 1917. efuse_interface_ctrl.EFUSE_PROGRAM_CTRL fields

Table 1918. efuse_interface_ctrl.EFUSE_READ_CTRL fields

Table 1919. efuse_interface_ctrl.EFUSE_PROGRAM_INTERFACE_READ_DATA fields

Table 1920. efuse_interface_ctrl.EFUSE_READ_INTERFACE_READ_DATA fields

Table 1921. efuse_interface_ctrl.EFUSE_READ_REQ_TIMEOUT fields

Table 1922. efuse_interface_ctrl.EFUSE_PROGRAM_REQ_TIMEOUT fields

Table 1923. eFuse SHIM Control Model Address Map register list

Table 1924. efuse_shim_ctrl.EFUSE_BANK_INIT_TIME fields

Table 1925. UART serial errors

Table 1926. UART interrupt priority and service

Table 1927. UART 16550 Main Address Map register list

Table 1928. uart_16550_main.RBR fields

Table 1929. uart_16550_main.IER fields

Table 1930. uart_16550_main.IIR fields

Table 1931. uart_16550_main.LCR fields

Table 1932. uart_16550_main.MCR fields

Table 1933. uart_16550_main.LSR fields

Table 1934. uart_16550_main.MSR fields

Table 1935. uart_16550_main.SCR fields

Table 1936. uart_16550_main.ECR fields

Table 1937. uart_16550_main.ITR fields

Table 1938. UART 16550 Main Write-Only Address Map register list

Table 1939. uart_16550_main_wo.THR fields

Table 1940. uart_16550_main_wo.FCR fields

Table 1941. UART 16550 Divisor Latch Address Map register list

Table 1942. uart_16550_dl.DLL fields

Table 1943. uart_16550_dl.DLM fields

Table 1944. Log-engine configuration

Table 1945. Log-engine register summary

Table 1946. Log Engine Address Map register list

Table 1947. log_engine.CTRL fields

Table 1948. log_engine.LOG_REGION_SIZE fields

Table 1949. log_engine.LOG_REGION_ADDR fields

Table 1950. log_engine.LOG_WRITE_ADDR fields

Table 1951. log_engine.INTR_STATUS fields

Table 1952. log_engine.INTR_ENABLE fields

Table 1953. log_engine.INTR_TEST fields

Table 1954. log_engine.LOG_CTRL[16] fields

Table 1955. I2C Address Map register list

Table 1956. i2c.INTR_STATE fields

Table 1957. i2c.INTR_ENABLE fields

Table 1958. i2c.INTR_TEST fields

Table 1959. i2c.SMBUS_CTRL fields

Table 1960. i2c.CTRL fields

Table 1961. i2c.STATUS fields

Table 1962. i2c.RDATA fields

Table 1963. i2c.FDATA fields

Table 1964. i2c.FIFO_CTRL fields

Table 1965. i2c.HOST_FIFO_CONFIG fields

Table 1966. i2c.TARGET_FIFO_CONFIG fields

Table 1967. i2c.HOST_FIFO_STATUS fields

Table 1968. i2c.TARGET_FIFO_STATUS fields

Table 1969. i2c.OVRD fields

Table 1970. i2c.VAL fields

Table 1971. i2c.TIMING0 fields

Table 1972. i2c.TIMING1 fields

Table 1973. i2c.TIMING2 fields

Table 1974. i2c.TIMING3 fields

Table 1975. i2c.TIMING4 fields

Table 1976. i2c.TIMEOUT_CTRL fields

Table 1977. i2c.TARGET_ID fields

Table 1978. i2c.ACQDATA fields

Table 1979. i2c.TXDATA fields

Table 1980. i2c.HOST_TIMEOUT_CTRL fields

Table 1981. i2c.TARGET_TIMEOUT_CTRL fields

Table 1982. i2c.TARGET_NACK_COUNT fields

Table 1983. i2c.TARGET_ACK_CTRL fields

Table 1984. i2c.ACQ_FIFO_NEXT_DATA fields

Table 1985. i2c.HOST_NACK_HANDLER_TIMEOUT fields

Table 1986. i2c.CONTROLLER_EVENTS fields

Table 1987. i2c.TARGET_EVENTS fields

Table 1988. i2c.SMBUS_STATUS fields

Table 1989. I2C Control Address Map register list

Table 1990. i2c_ctrl.I2C_CTRL[3] fields

Table 1991. I2C Wrapper Address Map register list

Table 1992. i2c_wrap.i2c[3].INTR_STATE fields

Table 1993. i2c_wrap.i2c[3].INTR_ENABLE fields

Table 1994. i2c_wrap.i2c[3].INTR_TEST fields

Table 1995. i2c_wrap.i2c[3].SMBUS_CTRL fields

Table 1996. i2c_wrap.i2c[3].CTRL fields

Table 1997. i2c_wrap.i2c[3].STATUS fields

Table 1998. i2c_wrap.i2c[3].RDATA fields

Table 1999. i2c_wrap.i2c[3].FDATA fields

Table 2000. i2c_wrap.i2c[3].FIFO_CTRL fields

Table 2001. i2c_wrap.i2c[3].HOST_FIFO_CONFIG fields

Table 2002. i2c_wrap.i2c[3].TARGET_FIFO_CONFIG fields

Table 2003. i2c_wrap.i2c[3].HOST_FIFO_STATUS fields

Table 2004. i2c_wrap.i2c[3].TARGET_FIFO_STATUS fields

Table 2005. i2c_wrap.i2c[3].OVRD fields

Table 2006. i2c_wrap.i2c[3].VAL fields

Table 2007. i2c_wrap.i2c[3].TIMING0 fields

Table 2008. i2c_wrap.i2c[3].TIMING1 fields

Table 2009. i2c_wrap.i2c[3].TIMING2 fields

Table 2010. i2c_wrap.i2c[3].TIMING3 fields

Table 2011. i2c_wrap.i2c[3].TIMING4 fields

Table 2012. i2c_wrap.i2c[3].TIMEOUT_CTRL fields

Table 2013. i2c_wrap.i2c[3].TARGET_ID fields

Table 2014. i2c_wrap.i2c[3].ACQDATA fields

Table 2015. i2c_wrap.i2c[3].TXDATA fields

Table 2016. i2c_wrap.i2c[3].HOST_TIMEOUT_CTRL fields

Table 2017. i2c_wrap.i2c[3].TARGET_TIMEOUT_CTRL fields

Table 2018. i2c_wrap.i2c[3].TARGET_NACK_COUNT fields

Table 2019. i2c_wrap.i2c[3].TARGET_ACK_CTRL fields

Table 2020. i2c_wrap.i2c[3].ACQ_FIFO_NEXT_DATA fields

Table 2021. i2c_wrap.i2c[3].HOST_NACK_HANDLER_TIMEOUT fields

Table 2022. i2c_wrap.i2c[3].CONTROLLER_EVENTS fields

Table 2023. i2c_wrap.i2c[3].TARGET_EVENTS fields

Table 2024. i2c_wrap.i2c[3].SMBUS_STATUS fields

Table 2025. i2c_wrap.I2C_CTRL[3] fields

Table 2026. OCA I3C Controller Wrapper Address Map register list

Table 2027. oca_i3c_wrap.i3c_csr[6].I3CBase.HCI_VERSION fields

Table 2028. oca_i3c_wrap.i3c_csr[6].I3CBase.HC_CONTROL fields

Table 2029. oca_i3c_wrap.i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR fields

Table 2030. oca_i3c_wrap.i3c_csr[6].I3CBase.HC_CAPABILITIES fields

Table 2031. oca_i3c_wrap.i3c_csr[6].I3CBase.RESET_CONTROL fields

Table 2032. oca_i3c_wrap.i3c_csr[6].I3CBase.PRESENT_STATE fields

Table 2033. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_STATUS fields

Table 2034. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_STATUS_ENABLE fields

Table 2035. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE fields

Table 2036. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_FORCE fields

Table 2037. oca_i3c_wrap.i3c_csr[6].I3CBase.DAT_SECTION_OFFSET fields

Table 2038. oca_i3c_wrap.i3c_csr[6].I3CBase.DCT_SECTION_OFFSET fields

Table 2039. oca_i3c_wrap.i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET fields

Table 2040. oca_i3c_wrap.i3c_csr[6].I3CBase.PIO_SECTION_OFFSET fields

Table 2041. oca_i3c_wrap.i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET fields

Table 2042. oca_i3c_wrap.i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN fields

Table 2043. oca_i3c_wrap.i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL fields

Table 2044. oca_i3c_wrap.i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL fields

Table 2045. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_BASE_LO fields

Table 2046. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_BASE_HI fields

Table 2047. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_SG fields

Table 2048. oca_i3c_wrap.i3c_csr[6].PIOControl.COMMAND_PORT fields

Table 2049. oca_i3c_wrap.i3c_csr[6].PIOControl.RESPONSE_PORT fields

Table 2050. oca_i3c_wrap.i3c_csr[6].PIOControl.TX_DATA_PORT fields

Table 2051. oca_i3c_wrap.i3c_csr[6].PIOControl.RX_DATA_PORT fields

Table 2052. oca_i3c_wrap.i3c_csr[6].PIOControl.IBI_PORT fields

Table 2053. oca_i3c_wrap.i3c_csr[6].PIOControl.QUEUE_THLD_CTRL fields

Table 2054. oca_i3c_wrap.i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL fields

Table 2055. oca_i3c_wrap.i3c_csr[6].PIOControl.QUEUE_SIZE fields

Table 2056. oca_i3c_wrap.i3c_csr[6].PIOControl.ALT_QUEUE_SIZE fields

Table 2057. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_STATUS fields

Table 2058. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE fields

Table 2059. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE fields

Table 2060. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_FORCE fields

Table 2061. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_CONTROL fields

Table 2062. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER fields

Table 2063. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0 fields

Table 2064. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1 fields

Table 2065. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2 fields

Table 2066. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3 fields

Table 2067. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0 fields

Table 2068. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1 fields

Table 2069. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2 fields

Table 2070. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3 fields

Table 2071. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4 fields

Table 2072. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5 fields

Table 2073. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED fields

Table 2074. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0 fields

Table 2075. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1 fields

Table 2076. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET fields

Table 2077. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL fields

Table 2078. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS fields

Table 2079. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS fields

Table 2080. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0 fields

Table 2081. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1 fields

Table 2082. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0 fields

Table 2083. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1 fields

Table 2084. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2 fields

Table 2085. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3 fields

Table 2086. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4 fields

Table 2087. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED fields

Table 2088. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA fields

Table 2089. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER fields

Table 2090. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL fields

Table 2091. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR fields

Table 2092. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES fields

Table 2093. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR fields

Table 2094. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS fields

Table 2095. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR fields

Table 2096. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO fields

Table 2097. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS fields

Table 2098. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO fields

Table 2099. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE fields

Table 2100. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE fields

Table 2101. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS fields

Table 2102. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS fields

Table 2103. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR fields

Table 2104. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL fields

Table 2105. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL fields

Table 2106. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER fields

Table 2107. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.CONTROL fields

Table 2108. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.STATUS fields

Table 2109. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RESET_CONTROL fields

Table 2110. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS fields

Table 2111. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH fields

Table 2112. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH fields

Table 2113. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH fields

Table 2114. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS fields

Table 2115. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE fields

Table 2116. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE fields

Table 2117. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL fields

Table 2118. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS fields

Table 2119. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE fields

Table 2120. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE fields

Table 2121. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0 fields

Table 2122. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1 fields

Table 2123. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2 fields

Table 2124. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3 fields

Table 2125. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4 fields

Table 2126. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5 fields

Table 2127. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING fields

Table 2128. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC fields

Table 2129. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH fields

Table 2130. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY fields

Table 2131. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED fields

Table 2132. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW fields

Table 2133. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW fields

Table 2134. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT fields

Table 2135. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT fields

Table 2136. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT fields

Table 2137. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT fields

Table 2138. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_PORT fields

Table 2139. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE fields

Table 2140. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE fields

Table 2141. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL fields

Table 2142. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL fields

Table 2143. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER fields

Table 2144. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL fields

Table 2145. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS fields

Table 2146. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG fields

Table 2147. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS fields

Table 2148. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2 fields

Table 2149. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3 fields

Table 2150. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF fields

Table 2151. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR fields

Table 2152. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2 fields

Table 2153. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3 fields

Table 2154. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG fields

Table 2155. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG fields

Table 2156. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG fields

Table 2157. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG fields

Table 2158. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG fields

Table 2159. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG fields
Table 2160. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG fields
Table 2161. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG fields
Table 2162. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG fields
Table 2163. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG fields
Table 2164. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG fields
Table 2165. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG fields
Table 2166. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG fields
Table 2167. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG fields
Table 2168. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG fields
Table 2169. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG fields
Table 2170. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG fields
Table 2171. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG fields
Table 2172. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG fields
Table 2173. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG fields
Table 2174. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG fields
Table 2175. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG fields
Table 2176. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG fields
Table 2177. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG fields
Table 2178. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG fields
Table 2179. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG fields
Table 2180. oca_i3c_wrap.i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER fields
Table 2181. oca_i3c_wrap.i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG fields
Table 2182. oca_i3c_wrap.i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER fields
Table 2183. oca_i3c_wrap.i3c_csr[6].DAT.DAT_MEMORY[128] fields
Table 2184. oca_i3c_wrap.i3c_csr[6].DCT.DCT_MEMORY[128] fields
Table 2185. AVSBus clock sources
Table 2186. AVSBus command and readback FIFOs
Table 2187. AVSBus protocol state encodings
Table 2188. AVSBus error handling
Table 2189. AVSBus Master Subframe
Table 2190. AVSBus Command Codes (Command Group 0)
Table 2191. AVSBus Slave Subframe
Table 2192. AVS_CLOCK_SELECT settings
Table 2193. AVS Clock Configuration Fields
Table 2194. AVSBus Reset Domains

Table 2195. AVSBus Interrupt Sources

Table 2196. AVSBus Protocol Timing

Table 2197. AVSBus Controller Address Map register list

Table 2198. avsbus_controller.AVS_CMD fields

Table 2199. avsbus_controller.AVS_READBACK fields

Table 2200. avsbus_controller.AVS_DEBUG_READBACK fields

Table 2201. avsbus_controller.AVS_LATEST_SLAVE_SUBFRAME fields

Table 2202. avsbus_controller.AVS_NORMAL_STATUS fields

Table 2203. avsbus_controller.AVS_SLAVE_STATUS fields

Table 2204. avsbus_controller.AVS_FIFOS_STATUS fields

Table 2205. avsbus_controller.AVS_INTERRUPT fields

Table 2206. avsbus_controller.AVS_INTERRUPT_MASK fields

Table 2207. avsbus_controller.AVS_INTERRUPT_CLEAR fields

Table 2208. avsbus_controller.AVS_CFG_0 fields

Table 2209. avsbus_controller.AVS_CFG_1 fields

Table 2210. avsbus_controller.AVS_CONFIG fields

Table 2211. GPIO data and direction selection

Table 2212. GPIO interrupt modes

Table 2213. GPIO Interface Address Map register list

Table 2214. gpio_intf.DATA_CTRL fields

Table 2215. gpio_intf.DATA_CTRL_ENABLE fields

Table 2216. gpio_intf.ACCESS_FILTER fields

Table 2217. GPIO Control Address Map register list

Table 2218. gpio_ctrl.CONTROL fields

Table 2219. GPIO Wrapper Address Map register list

Table 2220. gpio_wrap.DATA_CTRL fields

Table 2221. gpio_wrap.DATA_CTRL_ENABLE fields

Table 2222. gpio_wrap.ACCESS_FILTER fields

Table 2223. gpio_wrap.CONTROL fields

Table 2224. Telemetry packet fields

Table 2225. Telemetry interrupt conditions

Table 2226. Telemetry Receiver Address Map register list

Table 2227. telemetry_receiver.CTRL fields

Table 2228. telemetry_receiver.STATUS fields

Table 2229. telemetry_receiver.INTR_STATUS fields

Table 2230. telemetry_receiver.INTR_ENABLE fields

Table 2231. telemetry_receiver.INTR_TEST fields

Table 2232. telemetry_receiver.TELEMETRY_PROBE_ID fields

Table 2233. telemetry_receiver.TELEMETRY_COUNTER_VLDS fields

Table 2234. telemetry_receiver.TELEMETRY_COUNTER[32] fields

Table 2235. Telemetry Receiver Wrapper Address Map register list

Table 2236. telemetry_receiver_wrap.telemetry_receiver[3].CTRL fields

Table 2237. telemetry_receiver_wrap.telemetry_receiver[3].STATUS fields

Table 2238. telemetry_receiver_wrap.telemetry_receiver[3].INTR_STATUS fields

Table 2239. telemetry_receiver_wrap.telemetry_receiver[3].INTR_ENABLE fields

Table 2240. telemetry_receiver_wrap.telemetry_receiver[3].INTR_TEST fields

Table 2241. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_PROBE_ID fields

Table 2242. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_COUNTER_VLDS fields

Table 2243. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_COUNTER[32] fields

Table 2244. UART Wrapper Address Map register list

Table 2245. uart_wrap.uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL fields

Table 2246. uart_wrap.uart_log_engine_wrap[4].uart.RBR fields

Table 2247. uart_wrap.uart_log_engine_wrap[4].uart.IER fields

Table 2248. uart_wrap.uart_log_engine_wrap[4].uart.IIR fields

Table 2249. uart_wrap.uart_log_engine_wrap[4].uart.LCR fields

Table 2250. uart_wrap.uart_log_engine_wrap[4].uart.MCR fields

Table 2251. uart_wrap.uart_log_engine_wrap[4].uart.LSR fields

Table 2252. uart_wrap.uart_log_engine_wrap[4].uart.MSR fields

Table 2253. uart_wrap.uart_log_engine_wrap[4].uart.SCR fields

Table 2254. uart_wrap.uart_log_engine_wrap[4].uart.ECR fields

Table 2255. uart_wrap.uart_log_engine_wrap[4].uart.ITR fields

Table 2256. uart_wrap.uart_log_engine_wrap[4].log_engine.CTRL fields

Table 2257. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE fields

Table 2258. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR fields

Table 2259. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR fields

Table 2260. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_STATUS fields

Table 2261. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_ENABLE fields

Table 2262. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_TEST fields

Table 2263. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_CTRL[16] fields

Table 2264. UART Log Engine Control Address Map register list

Table 2265. uart_log_engine_ctrl.CTRL fields

Table 2266. UART Log Engine Wrapper Address Map register list

Table 2267. uart_log_engine_wrap.uart_log_engine_ctrl.CTRL fields
Table 2268. uart_log_engine_wrap.RBR fields
Table 2269. uart_log_engine_wrap.IER fields
Table 2270. uart_log_engine_wrap.IIR fields
Table 2271. uart_log_engine_wrap.LCR fields
Table 2272. uart_log_engine_wrap.MCR fields
Table 2273. uart_log_engine_wrap.LSR fields
Table 2274. uart_log_engine_wrap.MSR fields
Table 2275. uart_log_engine_wrap.SCR fields
Table 2276. uart_log_engine_wrap.ECR fields
Table 2277. uart_log_engine_wrap.ITR fields
Table 2278. uart_log_engine_wrap.log_engine.CTRL fields
Table 2279. uart_log_engine_wrap.LOG_REGION_SIZE fields
Table 2280. uart_log_engine_wrap.LOG_REGION_ADDR fields
Table 2281. uart_log_engine_wrap.LOG_WRITE_ADDR fields
Table 2282. uart_log_engine_wrap.INTR_STATUS fields
Table 2283. uart_log_engine_wrap.INTR_ENABLE fields
Table 2284. uart_log_engine_wrap.INTR_TEST fields
Table 2285. uart_log_engine_wrap.LOG_CTRL[16] fields
Table 2286. SMC Peripheral Parameter Overrides
Table 2287. SMC CLA event types
Table 2288. SMC DFD MMR Aperture Address Map register list
Table 2289. smc_cla.Trdstramcontrol fields
Table 2290. smc_cla.Trdstramimpl fields
Table 2291. smc_cla.Trdstramstartlow fields
Table 2292. smc_cla.Trdstramstarthigh fields
Table 2293. smc_cla.Trdstramlimitlow fields
Table 2294. smc_cla.Trdstramlimithigh fields
Table 2295. smc_cla.Trdstramwplow fields
Table 2296. smc_cla.Trdstramwphigh fields
Table 2297. smc_cla.Trdstramrplow fields
Table 2298. smc_cla.Trdstramrphigh fields
Table 2299. smc_cla.Trdstramdata fields
Table 2300. smc_cla.dst_sink.ScratchLo fields
Table 2301. smc_cla.dst_sink.ScratchHi fields
Table 2302. smc_cla.Trfunnelcontrol fields

Table 2303. smc_cla.Trfunnelimpl fields

Table 2304. smc_cla.Trfunneldisinput fields

Table 2305. smc_cla.funnel.ScratchLo fields

Table 2306. smc_cla.funnel.ScratchHi fields

Table 2307. smc_cla.CDbgClaCounter0Cfg fields

Table 2308. smc_cla.CDbgClaCounter1Cfg fields

Table 2309. smc_cla.CDbgClaCounter2Cfg fields

Table 2310. smc_cla.CDbgClaCounter3Cfg fields

Table 2311. smc_cla.CDbgNode0Eap0 fields

Table 2312. smc_cla.CDbgNode0Eap1 fields

Table 2313. smc_cla.CDbgNode1Eap0 fields

Table 2314. smc_cla.CDbgNode1Eap1 fields

Table 2315. smc_cla.CDbgNode2Eap0 fields

Table 2316. smc_cla.CDbgNode2Eap1 fields

Table 2317. smc_cla.CDbgNode3Eap0 fields

Table 2318. smc_cla.CDbgNode3Eap1 fields

Table 2319. smc_cla.CDbgSignalMask0Lo fields

Table 2320. smc_cla.CDbgSignalMatch0Lo fields

Table 2321. smc_cla.CDbgSignalMask1Lo fields

Table 2322. smc_cla.CDbgSignalMatch1Lo fields

Table 2323. smc_cla.CDbgSignalEdgeDetectCfg fields

Table 2324. smc_cla.CDbgEapStatus fields

Table 2325. smc_cla.CDbgClaCtrlStatus fields

Table 2326. smc_cla.CDbgMuxSelLo fields

Table 2327. smc_cla.CDbgRsvd1 fields

Table 2328. smc_cla.CDbgRsvd2 fields

Table 2329. smc_cla.CDbgTransitionMaskLo fields

Table 2330. smc_cla.CDbgTransitionFromValueLo fields

Table 2331. smc_cla.CDbgTransitionToValueLo fields

Table 2332. smc_cla.CDbgOnesCountMaskLo fields

Table 2333. smc_cla.CDbgOnesCountValue fields

Table 2334. smc_cla.CDbgAnyChangeLo fields

Table 2335. smc_cla.CDbgSignalSnapshotNode0Eap0Lo fields

Table 2336. smc_cla.CDbgSignalSnapshotNode0Eap1Lo fields

Table 2337. smc_cla.CDbgSignalSnapshotNode1Eap0Lo fields

Table 2338. smc_cla.CDbgSignalSnapshotNode1Eap1Lo fields

Table 2339. smc_cla.CDbgSignalSnapshotNode2Eap0Lo fields
Table 2340. smc_cla.CDbgSignalSnapshotNode2Eap1Lo fields
Table 2341. smc_cla.CDbgSignalSnapshotNode3Eap0Lo fields
Table 2342. smc_cla.CDbgSignalSnapshotNode3Eap1Lo fields
Table 2343. smc_cla.CDbgClaTimeMatch fields
Table 2344. smc_cla.CDbgSignalMask2Lo fields
Table 2345. smc_cla.CDbgSignalMatch2Lo fields
Table 2346. smc_cla.CDbgSignalMask3Lo fields
Table 2347. smc_cla.CDbgSignalMatch3Lo fields
Table 2348. smc_cla.CDbgNode0Eap2 fields
Table 2349. smc_cla.CDbgNode0Eap3 fields
Table 2350. smc_cla.CDbgNode1Eap2 fields
Table 2351. smc_cla.CDbgNode1Eap3 fields
Table 2352. smc_cla.CDbgNode2Eap2 fields
Table 2353. smc_cla.CDbgNode2Eap3 fields
Table 2354. smc_cla.CDbgNode3Eap2 fields
Table 2355. smc_cla.CDbgNode3Eap3 fields
Table 2356. smc_cla.CDbgSignalSnapshotNode0Eap2Lo fields
Table 2357. smc_cla.CDbgSignalSnapshotNode0Eap3Lo fields
Table 2358. smc_cla.CDbgSignalSnapshotNode1Eap2Lo fields
Table 2359. smc_cla.CDbgSignalSnapshotNode1Eap3Lo fields
Table 2360. smc_cla.CDbgSignalSnapshotNode2Eap2Lo fields
Table 2361. smc_cla.CDbgSignalSnapshotNode2Eap3Lo fields
Table 2362. smc_cla.CDbgSignalSnapshotNode3Eap2Lo fields
Table 2363. smc_cla.CDbgSignalSnapshotNode3Eap3Lo fields
Table 2364. smc_cla.CDbgSignalDelayMuxSel fields
Table 2365. smc_cla.CDbgClaXtriggerTimestretch fields
Table 2366. smc_cla.CDbgClaTimestamp fields
Table 2367. smc_cla.CDbgClaTimestampSync fields
Table 2368. smc_cla.CDbgClaTimestampConfig fields
Table 2369. smc_cla.CDbgClaTimestampOffset fields
Table 2370. smc_cla.CDbgSignalMask0Hi fields
Table 2371. smc_cla.CDbgSignalMatch0Hi fields
Table 2372. smc_cla.CDbgSignalMask1Hi fields
Table 2373. smc_cla.CDbgSignalMatch1Hi fields
Table 2374. smc_cla.CDbgSignalMask2Hi fields

Table 2375. smc_cla.CDbgSignalMatch2Hi fields
Table 2376. smc_cla.CDbgSignalMask3Hi fields
Table 2377. smc_cla.CDbgSignalMatch3Hi fields
Table 2378. smc_cla.CDbgMuxSelHi fields
Table 2379. smc_cla.CDbgTransitionMaskHi fields
Table 2380. smc_cla.CDbgTransitionFromValueHi fields
Table 2381. smc_cla.CDbgTransitionToValueHi fields
Table 2382. smc_cla.CDbgOnesCountMaskHi fields
Table 2383. smc_cla.CDbgAnyChangeHi fields
Table 2384. smc_cla.CDbgSignalSnapshotNode0Eap0Hi fields
Table 2385. smc_cla.CDbgSignalSnapshotNode0Eap1Hi fields
Table 2386. smc_cla.CDbgSignalSnapshotNode1Eap0Hi fields
Table 2387. smc_cla.CDbgSignalSnapshotNode1Eap1Hi fields
Table 2388. smc_cla.CDbgSignalSnapshotNode2Eap0Hi fields
Table 2389. smc_cla.CDbgSignalSnapshotNode2Eap1Hi fields
Table 2390. smc_cla.CDbgSignalSnapshotNode3Eap0Hi fields
Table 2391. smc_cla.CDbgSignalSnapshotNode3Eap1Hi fields
Table 2392. smc_cla.CDbgSignalSnapshotNode0Eap2Hi fields
Table 2393. smc_cla.CDbgSignalSnapshotNode0Eap3Hi fields
Table 2394. smc_cla.CDbgSignalSnapshotNode1Eap2Hi fields
Table 2395. smc_cla.CDbgSignalSnapshotNode1Eap3Hi fields
Table 2396. smc_cla.CDbgSignalSnapshotNode2Eap2Hi fields
Table 2397. smc_cla.CDbgSignalSnapshotNode2Eap3Hi fields
Table 2398. smc_cla.CDbgSignalSnapshotNode3Eap2Hi fields
Table 2399. smc_cla.CDbgSignalSnapshotNode3Eap3Hi fields
Table 2400. smc_cla.CDbgLfsr fields
Table 2401. smc_cla.CDbgLfsrMask fields
Table 2402. smc_cla.CDbgTimestampCapture fields
Table 2403. smc_cla.CDbgCompare0Lo fields
Table 2404. smc_cla.CDbgCompare0MaskLo fields
Table 2405. smc_cla.CDbgCompare1Lo fields
Table 2406. smc_cla.CDbgCompare1MaskLo fields
Table 2407. smc_cla.CDbgCompare2Lo fields
Table 2408. smc_cla.CDbgCompare2MaskLo fields
Table 2409. smc_cla.CDbgCompare3Lo fields
Table 2410. smc_cla.CDbgCompare3MaskLo fields

Table 2411. smc_cla.CDbgCompare0Hi fields

Table 2412. smc_cla.CDbgCompare0MaskHi fields

Table 2413. smc_cla.CDbgCompare1Hi fields

Table 2414. smc_cla.CDbgCompare1MaskHi fields

Table 2415. smc_cla.CDbgCompare2Hi fields

Table 2416. smc_cla.CDbgCompare2MaskHi fields

Table 2417. smc_cla.CDbgCompare3Hi fields

Table 2418. smc_cla.CDbgCompare3MaskHi fields

Table 2419. smc_cla.Scratch fields

Table 2420. smc_cla.Trdstcontrol fields

Table 2421. smc_cla.Trdstimpl fields

Table 2422. smc_cla.Trdstinstfeatures fields

Table 2423. smc_cla.CDbgDebugTraceCfg fields

Table 2424. smc_cla.dst.ScratchLo fields

Table 2425. smc_cla.dst.ScratchHi fields

Table 2426. SMC eFuse shadow register classification

Table 2427. SMC shadow word allocation

Table 2428. SMC Functional Address-Space Organization

Table 2429. SMC Component Address Map

Table 2430. SMC Register Interface Standards

Table 2431. SMC Access Control Mechanisms

Table 2432. PLL Wrapper Placeholder Address Map register list

Table 2433. pll_wrap.mem_array[954] fields

Table 2434. PVT Wrapper Placeholder Address Map register list

Table 2435. pvt_wrap.mem_array[594] fields

Table 2436. Straps Address Map register list

Table 2437. straps.STRAPS_LO fields

Table 2438. straps.STRAPS_HI fields

Table 2439. Chip Configuration Address Map register list

Table 2440. chip_config.VERSION_LO fields

Table 2441. chip_config.VERSION_HI fields

Table 2442. chip_config.CHIP_ID fields

Table 2443. chip_config.LC_STATE fields

Table 2444. Scratch Address Map register list

Table 2445. scratch.SCRATCH[8] fields

Table 2446. NDM Reset Control Address Map register list

Table 2447. ndm_reset.NDMRESET_REQUEST fields

Table 2448. ndm_reset.NDMRESET_PROCESS fields

Table 2449. ndm_reset.NDMRESET_CLUSTER_COUNT fields

Table 2450. SMC Miscellaneous Wrapper Address Map register list

Table 2451. misc_wrap.scratch_cold.SCRATCH[8] fields

Table 2452. misc_wrap.scratch_cold_warm.SCRATCH[8] fields

Table 2453. misc_wrap.VERSION_LO fields

Table 2454. misc_wrap.VERSION_HI fields

Table 2455. misc_wrap.CHIP_ID fields

Table 2456. misc_wrap.LC_STATE fields

Table 2457. misc_wrap.NDMRESET_REQUEST fields

Table 2458. misc_wrap.NDMRESET_PROCESS fields

Table 2459. misc_wrap.NDMRESET_CLUSTER_COUNT fields

Table 2460. SMC Base Configuration Address Map register list

Table 2461. smc_base_config.GLOBAL_BASE fields

Table 2462. smc_base_config.LOCAL_BASE fields

Table 2463. smc_base_config.REGION_SIZE fields

Table 2464. smc_base_config.CLOCK_GATE_CONTROL fields

Table 2465. smc_base_config.HANG_DET_SYS_AXI_CTRL fields

Table 2466. smc_base_config.HANG_DET_SYS_AXI_TIMEOUT_THRESHOLD fields

Table 2467. smc_base_config.HANG_DET_SEP_AXI_CTRL fields

Table 2468. smc_base_config.HANG_DET_SEP_AXI_TIMEOUT_THRESHOLD fields

Table 2469. smc_base_config.HANG_DET_DATA_ACCEL_CTRL fields

Table 2470. smc_base_config.HANG_DET_DATA_ACCEL_TIMEOUT_THRESHOLD fields

Table 2471. DFX Control and Status Address Map register list

Table 2472. dfx_ctrl_status.STATUS_SMU fields

Table 2473. dfx_ctrl_status.DEBUG_CTRL fields

Table 2474. dfx_ctrl_status.DEBUG_BUS_MUX fields

Table 2475. SMC Component Documentation

Table 2476. DTP clock domains

Table 2477. DTP reset signals

Table 2478. DTP JTAG2AXI bridges

Table 2479. DTP debug-disable paths

Table 2480. JIU STAP chain hierarchy

Table 2481. JIU debug-disable fields

Table 2482. JIU iJTAG DR with every SIB closed

Table 2483. JIU instruction encodings
Table 2484. PTAP module hierarchy
Table 2485. PTAP state encodings
Table 2486. PTAP reset hierarchy
Table 2487. PTAP IDCODE fields
Table 2488. PTAP DEBUG_CONTROL fields
Table 2489. PTAP JTAG_CAPS fields
Table 2490. PTAP TMP_STATUS fields
Table 2491. PTAP 3DCR fields
Table 2492. PTAP IC_RESET fields
Table 2493. PTAP JTAG2AXI capability fields
Table 2494. JTAG2AXI single-operation fields
Table 2495. JTAG2AXI series-control fields
Table 2496. JTAG2AXI incrementing data fields
Table 2497. JTAG2AXI non-incrementing data fields
Table 2498. JTAG2AXI data and error-status fields
Table 2499. PTAP debug-disable inputs
Table 2500. PTAP instruction encodings
Table 2501. STAP module hierarchy
Table 2502. STAP 3DCR fields
Table 2503. DTP STAP allocation
Table 2504. STAP debug-disable behavior
Table 2505. CTN address allocation
Table 2506. External and internal CTP comparison
Table 2507. Internal CTP wire-OR connections
Table 2508. Internal CTP point-to-point connections
Table 2509. CTN matrix port indices
Table 2510. Cross Trigger Network Address Map
Table 2511. Cross Trigger Matrix Address Map register list
Table 2512. cross_trigger_matrix.CT_SRC[26].CONFIG_0 fields
Table 2513. Cross Trigger Port Address Map register list
Table 2514. cross_trigger_port.CONFIG fields
Table 2515. cross_trigger_port.STATUS fields
Table 2516. cross_trigger_port.STRETCH_MULT fields
Table 2517. Cross Trigger Network Address Map register list
Table 2518. cross_trigger_network.CT_SRC[26].CONFIG_0 fields

Table 2519. cross_trigger_network.ctp[16].CONFIG fields

Table 2520. cross_trigger_network.ctp[16].STATUS fields

Table 2521. cross_trigger_network.ctp[16].STRETCH_MULT fields

Table 2522. CTP configuration registers

Table 2523. CTP submodules

Table 2524. CTP signal synchronization

Table 2525. CTP sender state transitions

Table 2526. CTP receiver state transitions

Table 2527. CTM configuration constants

Table 2528. CTM routing stages

Table 2529. CTM port indices

Table 2530. DTP Port Declaration

Table 2531. AoU width-adaptation paths

Table 2532. AoU burst-splitting limits

Table 2533. AoU burst-splitter SFR controls

Table 2534. AoU early-response tracking fields

Table 2535. AoU early-response SFR fields

Table 2536. AoU aggregation SFR fields

Table 2537. WriteData Credit Calculation

Table 2538. WriteDataFull Credit Calculation

Table 2539. ReadData Credit Calculation

Table 2540. AoU flit-cancel cases

Table 2541. Interrupt Sources

Table 2542. AOU_CORE Initial Latency (Cycles)

Table 2543. AOU_CORE Data Transfer Efficiency

Table 2544. AOU_CORE Area (measured at an FDI_CFG_SP_64B-equivalent configuration; numbers will scale up for FDI_CFG_*_128B configurations due to the 88 -> 140 R/W FIFO depth bump.)

Table 2545. Referenced Documents

Table 2546. Traffic filter instances

Table 2547. Granule widening examples

Table 2548. Register field access tokens

Table 2549. OCAH documentation references

Table 2550. OCA specification references

Table 2551. Interface and security standard references

Table 2552. External design documentation references

# Chapter 1. OCAH Architecture & Design

This section provides an overview of the OCAH architecture including its subsystem composition, power and clock domains, reset hierarchy, and memory map. It is organised into focused topic pages; subsequent chapters provide detailed microarchitectural documentation for each subsystem.

## 1.1. Platform Overview

### 1.1.1. OCAH High-Level Features

- **Security Processor (SEP)** — Root of Trust (RoT) with the highest security privilege. Provides a physically isolated RISC-V execution environment for secure boot, lifecycle control, device attestation, and hardware-accelerated cryptography (OTBN PKA, AES, HMAC-SHA2, KMAC-SHA3, TRNG, DRBG, Key Manager). A security hierarchy across chiplets enables SiP-wide security management. The inclusion of the SEP is an elaboration option, controlled by the SMU's `CFG.SEP` field. The system integrator selects which chiplets include an SEP as part of the SiP security architecture.
- **System Management Controller (SMC)** — Firmware-driven orchestration engine responsible for clock, voltage, and reset management, platform bootstrapping, system initialisation, and power/thermal management. Built around a RISC-V micro-controller with hardware accelerators (DMA, memory zeroer) where firmware alone cannot meet timing requirements. Inter-chiplet management hierarchy via I2C/I3C and OCMI channels.
- **Debug & Test Ports (DTP)** — Standardised debug and test access across heterogeneous, multi-vendor chiplet systems. Layered architecture separating physical transport from logical access, with cross-chiplet event triggering, synchronised clock control for trace, and coordinated resets.
- **System Management Unit (SMU)** — The composite module combining SEP, SMC, and DTP into a single instantiable subsystem.
- **System Management Network (SMN)** — Internal communication fabric (point-to-point or packetised) providing the management backbone for configuration, status reporting, and firmware-driven coordination between SMU, user logic, and D2D interfaces.
- **AXI-over-UCIe Bridge (AoU)** — Protocol bridge between the AXI-based internal fabric and the UCIe die-to-die physical interface, enabling inter-chiplet system management traffic and application-level transaction forwarding.
- **Application-Specific Logic** — OCAH treats the application layer as a black box: the harness services are agnostic to whether the application is a CPU complex, NoC, DSA, memory controller, or high-speed IO. The application logic is outside the scope of OCAH.
- **Safety Monitor (SMO)** — External entity that collects error signals and diagnostic logging from OCAH subsystems to implement fault reaction policies. The SMO is outside the scope of OCAH; its interface requirements are specified separately as part of OCAH's functional safety documentation.

### 1.1.2. Top Block Diagram

Figure 1 presents the OCAH top-level block diagram showing the relationship between the subsystems described above.

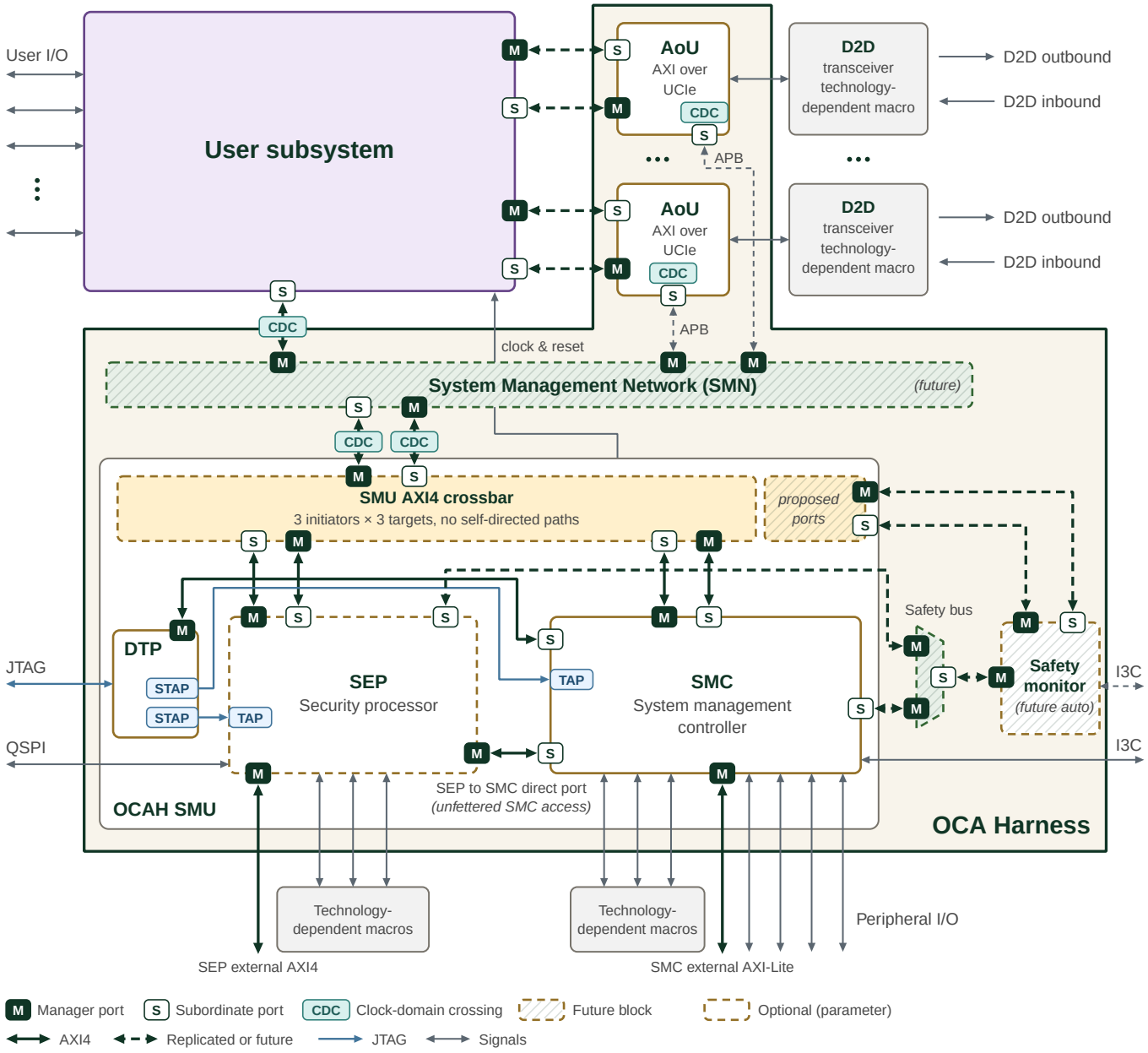


*Figure 1. OCAH Top-Level Block Diagram*

The SMU crossbar permits each initiator to reach the other two targets; self-directed paths are not connected. The detailed connectivity is shown in SMU AXI Crossbar.

Hatched blocks are future: the SMN, the safety bus and the safety monitor. The crossbar ports that would connect the safety monitor are proposed and are not part of the implemented crossbar. Dashed outlines without hatching mark the blocks present only when the SMU `CFG.SEP` field is 1: the SEP and the SMU crossbar.

# 1.2. OCAH Memory Map

OCAH separates the chiplet-visible SMU aperture from the internal address maps used by SEP and SMC firmware. The chiplet integrator decides which system address range reaches the SMU external AXI input. Firmware then programs the SEP and SMC global base/size CSRs so the SMU crossbar can route incoming traffic to the correct inbound port.

Table 1 distinguishes the chiplet envelope, programmable apertures, and internal address maps.

*Table 1. OCAH address-map configuration layers*

| Concept | Configured by | Purpose | Example |
|---|---|---|---|
| Chiplet NoC envelope | Integrator / chiplet address map | Controls which external-master addresses reach SMU `ext_in` | Example only: 64 MiB total SMU aperture |
| SEP global aperture | SEP firmware via `SEP_GLOBAL_BASE_ADDR` / `SEP_REGION_SIZE` | Controls SMU `ext_in` to `sep_in` routing and SEP inbound global-to-local remap | Example only: 32 MiB SEP slot in a 32 MiB + 32 MiB SEP/SMC partition |
| SMC global aperture | SMC firmware via `GLOBAL_BASE` / `REGION_SIZE` | Controls SMU `ext_in` to `smc_in` routing and SEP-to-SMC decode | Example only: 32 MiB SMC slot in a 32 MiB + 32 MiB SEP/SMC partition |

External masters use chiplet-global addresses and are bounded first by the chiplet NoC decode. The SMU crossbar applies the firmware-programmed SEP and SMC global apertures to route those transactions to `sep_in` or `smc_in`. Subsystem-internal CPU maps and local aliases are described in the SEP and SMC chapters.

At reset, `SEP_GLOBAL_BASE_ADDR` is `0x0000_0000` and the SMC's `GLOBAL_BASE` register is `0x4000_0000`. Their separate size registers, `SEP_REGION_SIZE` and SMC `REGION_SIZE`, both reset to `0x0100_0000` (16 MiB). The corresponding half-open address ranges are therefore `[0x0000_0000, 0x0100_0000)` for SEP and `[0x4000_0000, 0x4100_0000)` for SMC.

The global base addresses and region sizes are software-programmed and must not overlap. The hardware does not derive them from a chiplet ID. The Programmer's Guide gives the aperture programming sequence for early boot.

Register tables throughout this manual report each field's software access as a compact token in their **Access** column; Register Field Access Notation decodes that notation.

## 1.2.1. Decode Response Codes

The SMC, SEP, Key Manager and cross-trigger network address maps state, for each unit, how a 32-bit access that no register or memory backs is answered. A cell reads *read response, read data / write response*, for example `DECERR, 0xBADCAB1E / SLVERR`. `OKAY`, `SLVERR` and `DECERR` are the AXI response codes returned on `RRESP` and `BRESP`. The read data is the word a 32-bit read returns in either half of a 64-bit data bus. Where the two halves differ, a note on the cell gives the word returned when address bit 2 is set.

- The *Hole in Extent* column covers an unbacked offset inside the unit's decoded extent. Where the space between a wrapper's sub-blocks, or the padding after each element of an array, answers differently, the cell names it separately as `between sub-blocks` or `between instances`.
- The *Past Extent* column covers the rest of the unit's aperture, the range its Size column gives.
- A `Reserved` row is an address between unit apertures.

*Table 2. Decode response codes*

| Code | Meaning |
|---|---|
| `OKAY` | The access completes. A read returns the listed data, and a write has no effect. |
| `SLVERR` | The access reaches a component that refuses it. |
| `DECERR` | No unit or sub-block claims the address, and a decoder refuses the access before it reaches one. |
| Aliases registers | The access reaches a live register at another offset, which answers it. |
| Forwarded | The subsystem passes the access on without decoding it, and its target answers. |
| Adopter-defined | The adopter's implementation answers. |
| `-` | The region does not exist: the unit is memory, its registers fill its extent, or its aperture ends at its extent. |

## 1.3. Power Domains

The SMU, SMN, and AXI-over-UCIe bridge (AoU) reside in the same always-on power domain. This domain remains powered whenever the chiplet has supply, ensuring that platform management, inter-chiplet communication, and security services are available even when application logic is powered down.

Vendor-specific macros and subsystems that are outside the scope of OCAH — such as OTP/eFuse controllers, PLLs, die-to-die PHYs (e.g. UCIe PHY), analogue sensors, and application-specific tiles — typically reside in their own dedicated power and voltage domains defined by the adopter's power architecture. OCAH provides the isolation and sequencing infrastructure (`isolate_req_o[31:0]`, `ss_reset_ctrl_o[31:0]`) that the SMC firmware uses to coordinate power transitions for these external domains, but their supply rail definitions, power gating decisions, and leakage budgets are adopter-specific.

The SMC provides runtime power management services for the application logic, including dynamic voltage and frequency scaling (DVFS) through AVSBus-based voltage regulator control and firmware-driven Performance State (P-State) management. At the RTL level, the SMU accepts a `powergood_i` input that indicates supply stability and exposes `isolate_req_o[31:0]` to request isolation of individual subsystem domains before power transitions.

## 1.4. Clock Domains

OCAH distinguishes low-level management timing from functional application clocks. The SMC receives an always-on reference clock (`clk_ref_i`) that is available from power-on, before any PLL is locked. Using this reference clock, the SMC controls the PLLs during early bring-up and generates the primary system clock (`clk_smu_i`) that is then distributed to the SMC itself, the SEP, and the DTP.

The `clk_smn_i` input is architecturally independent of `clk_smu_i`; when they are asynchronous, CDCs at this boundary are an integration responsibility. OCAH does not supply synchronizer cells at the SMN interface.

Figure 2 overlays the clock domains on the OCAH top-level block diagram. Each zone boundary runs through the CDCs that separate the domains. The AoU bridges, the D2D macros and the future safety monitor are not assigned a zone. The SMN reaches each AoU through its APB register port, whose clock is asynchronous to the AoU core clock; the CDC marker inside each AoU is the bridge's own APB synchroniser. The SMU zone marks `clk_smu_i`, which clocks the DTP, the crossbar, the SEP and the SMC. Pills inside the SEP and SMC list their clock inputs: both also use `clk_ref_i`. The SMC uses `clk_periph_i` for its peripherals and `clk_telemetry_i` for ATB telemetry. The SEP uses `entropy_rosc_sample_clk_i`, shown as "entropy rosc", for its entropy source and `clk_sep_wdt_i` for its watchdog.

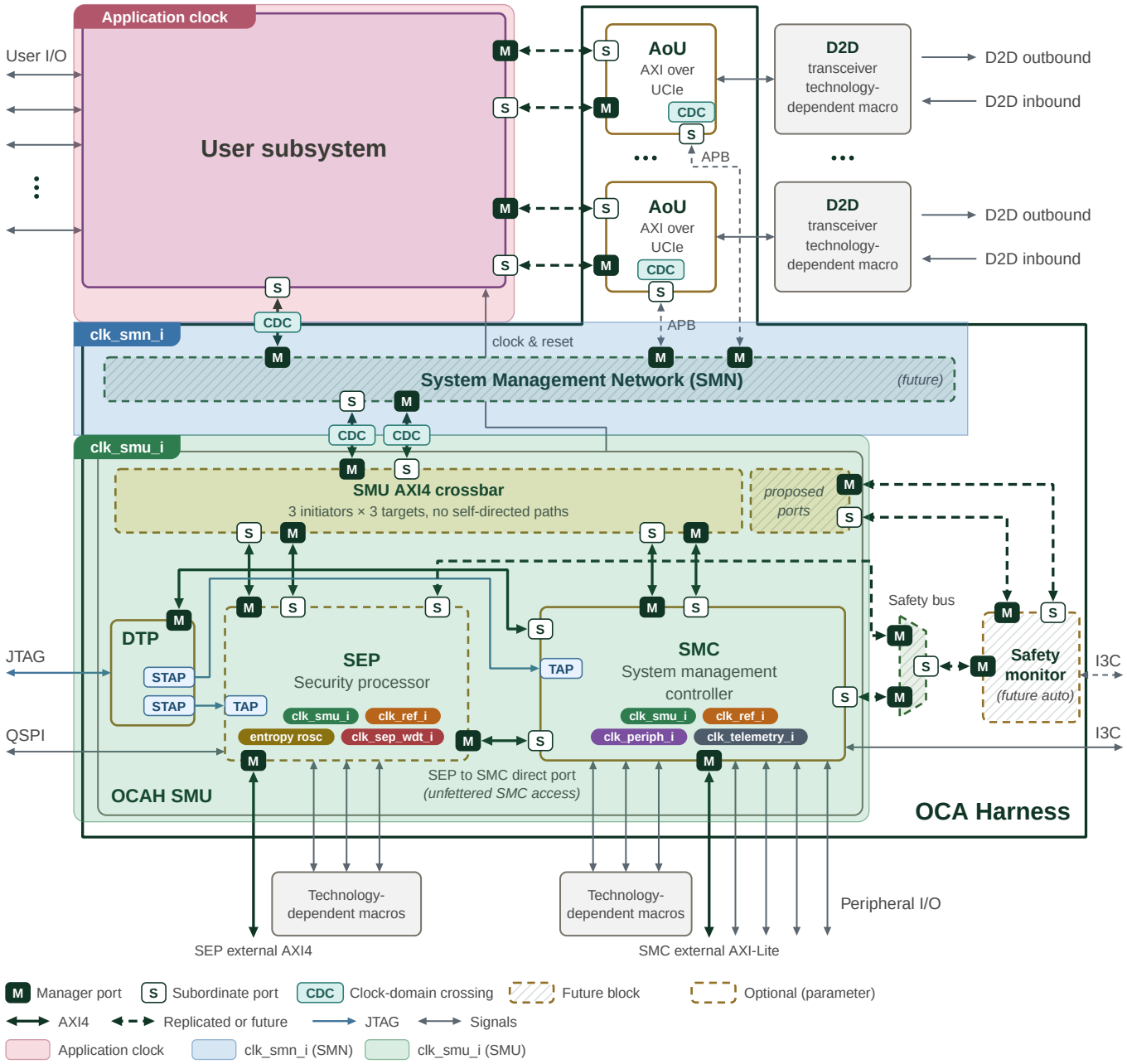


*Figure 2. OCAH Clock-Domain Zones*

Table 3 summarises the clock domains present in OCAH.

*Table 3. OCAH Clock Domains*

| Clock | Subsystem | Description |
|---|---|---|
| `clk_smu_i` | SMU (SMC, SEP, DTP) | Primary system clock, generated by the PLL under SMC control. Distributed internally to the SMC, SEP, and DTP subsystems. Drives the CPU cluster, fabric interconnect, peripheral controllers, and DTP cross-trigger / AXI bridges. |
| `clk_smn_i` | SMN (integration context) | System Management Network clock. An integration-level input independent of `clk_smu_i`; when the two are asynchronous, the integrator is responsible for placing CDCs at this boundary. OCAH does not supply synchronizer cells at the SMN interface. Not an SMU port. |

| Clock | Subsystem | Description |
|---|---|---|
| `clk_ref_i` | SMC, SEP | Always-on reference clock, available before PLL lock. Used by the SMC for early bring-up, PLL control, reset synchronisation, and peripherals requiring a stable, lower-frequency source. The SEP also clocks its reference counter from this clock. |
| `clk_periph_i` | SMC | Peripheral clock domain. Drives SMC peripheral subsystems. The reset unit generates a dedicated synchronised reset (`rst_primary_periph_clk`) in this domain. |
| `clk_telemetry_i` | SMC | ATB telemetry clock domain. Separate clock for Advanced Trace Bus data collection, with its own reset (`rst_telemetry_ni`). |
| `clk_sep_wdt_i` | SEP | Low-frequency watchdog timer clock. Must be independent of the system clock so the WDT can detect system clock failures. |
| `jtag_tck` | DTP / SEP | JTAG Test Clock. Drives PTAP, STAP state machines, and all JTAG-domain scan logic inside the DTP/SMC/SEP. |
| `entropy_rosc_sample_clk_i` | SEP | Ring-oscillator sample clock for the entropy source (TRNG). Must be asynchronous to `clk_i` and typically runs between 100 and 400 MHz. |

### 1.4.1. Reference Counter

Both the SMC and the SEP expose a free-running 64-bit reference counter in their CPU control register block (the `REFERENCE_COUNTER` CSR), clocked by the always-on reference clock (`clk_ref_i`). It advances continuously from reset, independent of the system clock (`clk_smu_i`) frequency or PLL state. Its value is synchronised into the CPU clock domain for software reads, and software may write the CSR to reload the counter, after which it resumes counting from the loaded value. Both subsystems share the same implementation (`prim_refclk_count_w_cdc`) and behave identically.

A reload crosses into the reference-clock domain through a one-entry asynchronous FIFO, and a second write issued before the first has crossed is dropped without an error indication. The Programmer's Guide gives the reload sequence.

## 1.5. Reset

Reset is handled hierarchically so that the management infrastructure comes out of reset first and then releases dependent domains in a controlled order. A global power-on reset (`rst_cold_ni`) establishes a known state for the entire chiplet, but the always-on domain is intended to be released first so that early control logic is live immediately. From there, the SMC sequences reset deassertion for the SMN and for downstream managed blocks according to platform dependencies, boot policy, and fault conditions. This staged reset model also supports partial and warm-reset behaviours in which application domains can be reset or recovered without fully tearing down the always-on management infrastructure.

The reset hierarchy within the SMU proceeds as follows:

1. **Power-on / cold reset.** The external `rst_cold_ni` is asserted during power-up and deasserted synchronously after supply voltages stabilise. The `powergood_i` input indicates that supplies are stable.
2. **SMC reset synchronisation.** The SMC reset unit (`smc_reset_unit`) generates domain-specific synchronised resets from the single cold reset input:
   - `rst_primary_smc_clk_no` — synchronised to the primary SMC clock domain
   - `rst_primary_ref_clk_no` — synchronised to the reference clock domain
   - `rst_primary_periph_clk_no` — synchronised to the peripheral clock domain
   - `rst_cold_stable_ref_clk_no` — stable cold reset in the reference domain
3. **DTP reset.** The DTP is held in reset by the SMC's primary synchronised reset (`rst_primary_smc_clk_no`). In addition,

the JTAG state machine requires a power-on reset that remains asserted until supplies are stable; at the SMU level this is the `powergood_stable` signal generated by the SMC.

4. **SEP reset.** The SEP is held in reset by the same SMC primary reset (`rst_primary_smc_clk_no`). The SEP watchdog timer also drives a shared warm-reset path (`rst_wdt_n` at the SMU level); its blast radius is limited to the SEP CPU and the SEP warm scratch registers, plus the parts of the SMC that the SMC reset unit routes through the warm-reset path.
5. **Downstream subsystem sequencing.** The SMC sequences reset deassertion for up to 32 managed subsystems through `ss_reset_ctrl_o[31:0]` and isolation through `isolate_req_o[31:0]`. Subsystems report completion via `ss_reset_complete_i[31:0]`.
6. **Function Level Reset (FLR).** Asserting `cfg_flr_pf_active_i` triggers a warm reset sequence that isolates subsystems through `isolate_req_o` and performs controlled reset propagation without tearing down the always-on infrastructure.
7. **Debug reset overrides.** The DTP provides JTAG IC_RESET controls that can override the fuse, cold, cool, warm, and per-subsystem reset domains independently for debug scenarios. These are wired through the SMU into the SMC reset unit and the SEP reset controller. The Programmer's Guide gives the programming order these overrides require.

### 1.5.1. IC_RESET TDR

The primary TAP implements the IEEE 1149.1-2013 §17 IC_RESET TDR as three slices on the scan chain from TDI to TDO, followed by a single `reset_hold` bit nearest TDO:

- **SMC slice** (nearest TDI). Fuse, cold, cool, and warm resets, plus cold and warm resets for each of the 32 managed subsystems. The SMU consumes this slice internally.
- **SEP slice.** SEP main reset and the Adams Bridge, TRNG, KMAC, HMAC, AES, OTBN, and Key Manager resets. Present when SEP is instantiated; omitted otherwise. The SMU consumes this slice internally.
- **External slice.** Reset overrides for logic outside the SMU. The ports in this slice are adopter-defined: the integrator supplies the slice type and routes the SMU output to that logic. OCAH does not name the external bits. The default type is a single-override stub.

Each port carries an override-enable bit and an active-low reset value. The Integrator Guide records the SMU port map, including how to size the external slice.

For the detailed reset controller design, sequencing FSM, reset register interface, and clock-reset interaction, refer to the Clock and Reset Management chapter:

Clock and Reset Management

For integration-level reset procedures, refer to the Integrator Guide. The Programmer's Guide lists the resets that software requests in its SMC and SEP reset overviews.

## 1.6. RTL Assertions

First-party RTL uses the `OCAH_ASSERT*` macros; OpenTitan-derived blocks use `OCAH_OT_ASSERT*`; PULP-vendored cells (`common_cells`, AXI, iDMA) use `OCAH_PULP_ASSERT*`. The checks live in the module that owns the behavior and cover known-value hygiene, illegal parameter combinations, and local protocol or FSM invariants. They compile out of synthesis and fire on four-state simulators.

Table 4 identifies the header for each assertion macro family.

*Table 4. RTL assertion macro families*

| Family | Header |
| --- | --- |
| `OCAH_ASSERT*` | `hw/common/assert/ocah_assert.svh` |
| `OCAH_OT_ASSERT*` | `vendor/lowRISC/opentitan/upstream/hw/ip/prim/rtl/prim_assert.sv` |
| `OCAH_PULP_ASSERT*` | `vendor/pulp-platform/common_cells/upstream/include/common_cells/assertions.svh` |

# Chapter 2. OCAH Platform Components

## 2.1. System Management Unit (SMU)

The System Management Unit (SMU) is the chiplet's system-management subsystem. It integrates three functional blocks around a runtime-programmable AXI crossbar and presents a single boundary to the System Management Network (SMN).

### 2.1.1. Composition

The SMU combines:

- **Security Processor (SEP)** — Root of Trust with a physically isolated RISC-V execution environment for secure boot, lifecycle control, device attestation and hardware-accelerated cryptography (OTBN PKA, AES, HMAC-SHA2, KMAC-SHA3, TRNG, DRBG, Key Manager). See [security-processor-sep].
- **System Management Controller (SMC)** — Firmware-driven orchestration engine responsible for clock, voltage and reset management, platform bootstrapping, system initialisation, and power/thermal management. Built on a RISC-V micro-controller with DMA and memory-zeroer accelerators. See [system-management-controller].
- **Debug and Test Ports (DTP)** — Standardised debug and test access across heterogeneous, multi-vendor chiplet systems. Covers JTAG scan chains, JTAG2AXI debug bridges, cross-trigger network and synchronised clock control. See Debug and Test Ports (DTP).

These three subsystems share a single primary clock domain (`clk_smu_i`) and are released from reset by the SMC's reset unit in a controlled sequence. Platform-wide clock, power and reset behaviour is described in the Clock Domains chapter.

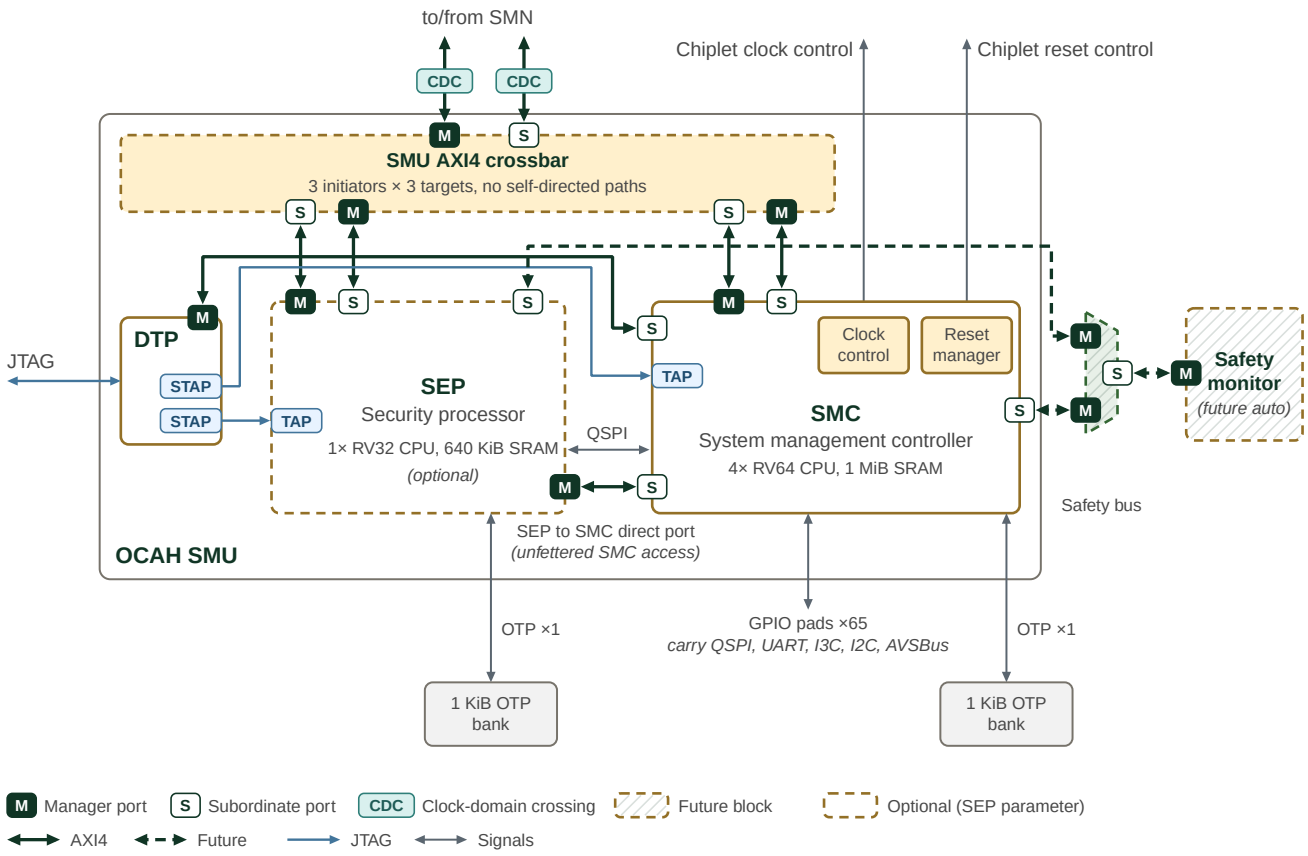


*Figure 3. SMU composition with the optional SEP subsystem*

The SMU composition is shown in Figure 3. Dashed outlines mark the blocks present only when `CFG.SEP` is 1: the SEP and the AXI4 crossbar. Memory capacities in the diagram describe the illustrated configuration. Hatched blocks are future: the safety bus and the safety monitor, whose connections to the SEP and SMC are drawn dashed.

## 2.1.2. AXI Crossbar

Inside the SMU, the `smu_axi_xbar` module is the system-level arbitration fabric between SEP, SMC, and the rest of the chiplet (reached through the `smu_axi_in` / `smu_axi_out` boundary pair). It is a 3×3 AXI4 crossbar with non-reflexive connectivity: each master can reach the other two subordinates but never its own inbound port.

The SMU instantiates the crossbar only when `CFG.SEP` is 1. When `CFG.SEP` is 0, AXI ID-width converters connect the SMC system AXI ports to the `smu_axi_in` / `smu_axi_out` pair: every SMC-initiated transaction leaves through `smu_axi_out`, and every `smu_axi_in` transaction targets the SMC.

| Crossbar targets | Initiators | | |
|---|---|---|---|
| | SEP sep_out | SMC smc_out | Chiplet fabric ext_in |
| sep_in (SEP) | — | ● | ● |
| smc_in (SMC) | ● | — | ● |
| ext_out (catch-all) | ● | ● | — |

● Connected — Not connected

*Figure 4. SMU AXI crossbar connectivity*

Table 5 lists the permitted initiator-to-subordinate paths and their purposes.

*Table 5. SMU AXI crossbar ports*

| Master (from) | Subordinate (to) | Purpose |
|---|---|---|
| `sep_out` | `smc_in` | SEP-initiated transactions into SMC resources |
| `sep_out` | `ext_out` | SEP-initiated transactions leaving the SMU |
| `smc_out` | `sep_in` | SMC-initiated transactions into SEP resources |
| `smc_out` | `ext_out` | SMC-initiated transactions leaving the SMU |
| `ext_in` | `sep_in` | External transactions targeting SEP |
| `ext_in` | `smc_in` | External transactions targeting SMC |

Routing decisions use a runtime address map with two programmable rules, the SEP and SMC apertures. `ext_out` has no address rule; it is the default subordinate for `sep_out` and `smc_out` requests that match neither aperture. See OCAH Memory Map for the fixed-vs-programmable split and the reference envelope convention, and see the SMU AXI Crossbar Address Map section in the Integrator Guide for firmware programming order.

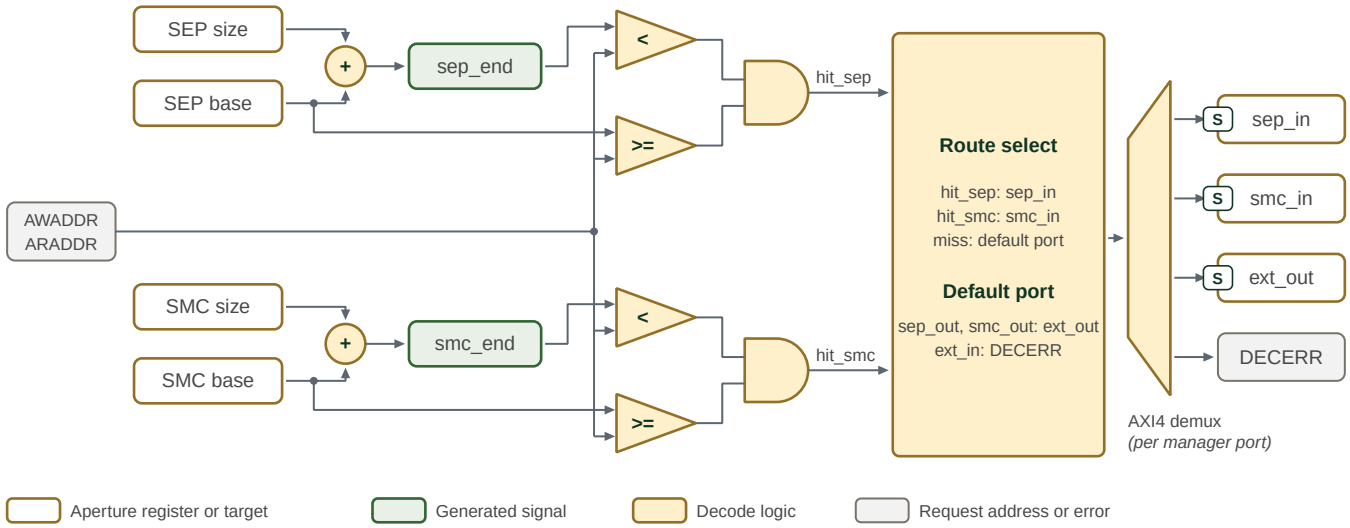


*Figure 5. SMU aperture decode and routing*

Figure 5 shows the independent SEP and SMC address comparisons for each read or write request. Each aperture includes its base address and excludes `base + size`; the addition uses 57 bits to preserve carry from the 56-bit address, and the address is zero-extended to 57 bits for the comparison. The SEP and SMC ranges must not overlap.

The decoded aperture and the connectivity in Figure 4 together determine where each request goes, as Table 6 shows.

*Table 6. SMU AXI crossbar routing by initiator and decoded aperture*

| Initiator | SEP aperture | SMC aperture | Neither aperture |
|---|---|---|---|
| `sep_out` | `DECERR` | `smc_in` | `ext_out` |
| `smc_out` | `sep_in` | `DECERR` | `ext_out` |
| `ext_in` | `sep_in` | `smc_in` | `DECERR` |

- A `sep_out` or `smc_out` request that matches neither aperture is forwarded to `ext_out`.
- A `sep_out` request that matches the SEP aperture, or an `smc_out` request that matches the SMC aperture, returns `DECERR`, because no initiator is connected to its own subsystem's inbound port.
- An `ext_in` request that matches neither aperture returns `DECERR`, because `ext_in` is not connected to `ext_out`.

The crossbar generates each `DECERR` response itself; the request reaches no subordinate.

### 2.1.3. Clocks and Resets

Figure 6 and Figure 7 show the clock and reset distribution from the SMU boundary to DTP, SEP, SMC, and the shared AXI infrastructure. Boundary inputs enter on the left and boundary outputs leave on the right; nets generated inside the SMU run in the channel to the right of the instances. A dot marks fan-out, and a gap in a line marks a crossing without a connection. Labels at an instance's right edge name the net it drives or the port that receives a generated net. Instance names match the SMU RTL hierarchy, and paths to the SEP exist only when `CFG.SEP` is 1.

Active-low resets use the `_ni`, `_no` or `trst_n` suffixes; the SMC CPU JTAG reset is active-high, so `dtp_smc_stap_tap_ctrl.trst_n` passes through an inverter. Figure 7 shows reset distribution within the SMU, not the reset logic inside each subsystem.

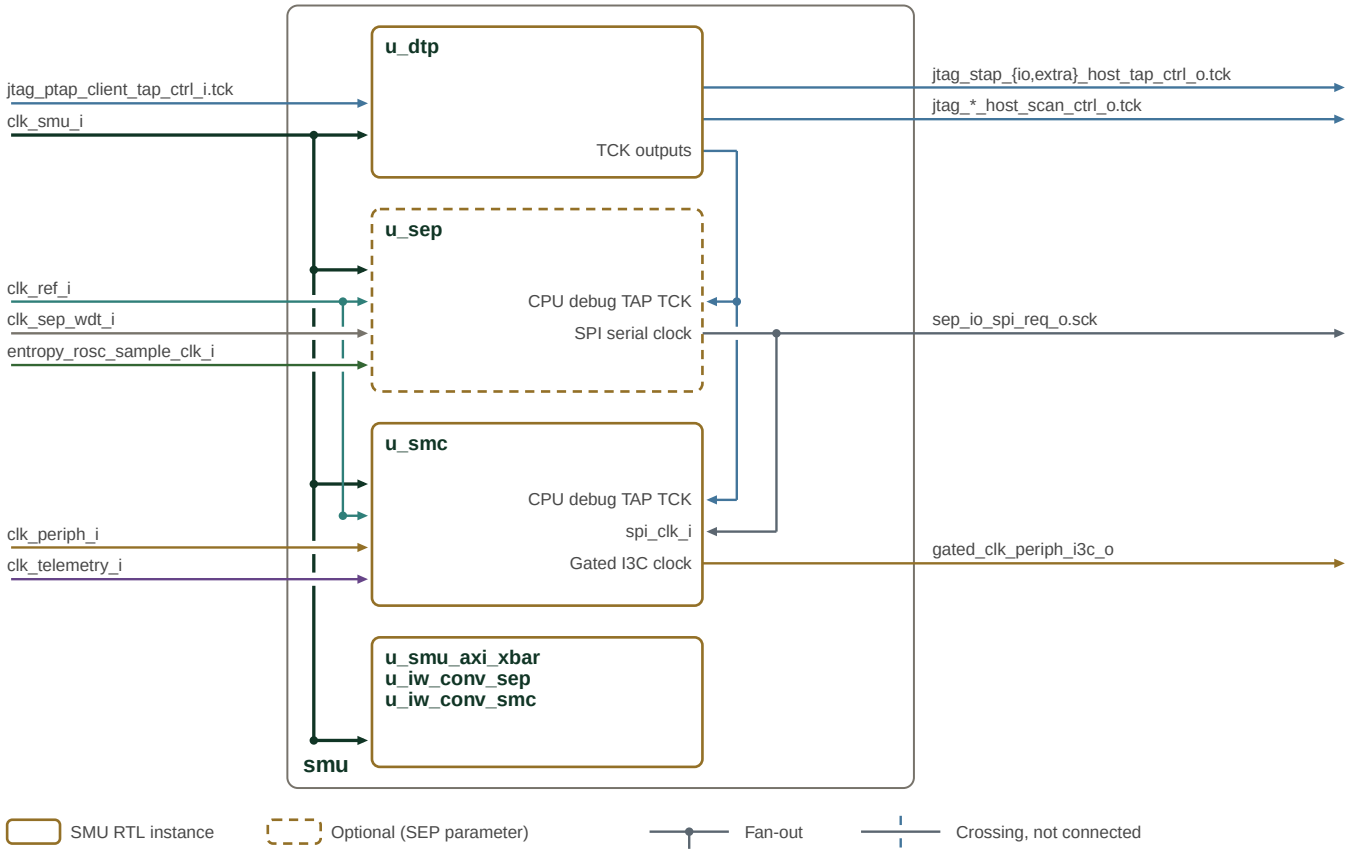


*Figure 6. SMU clock distribution*

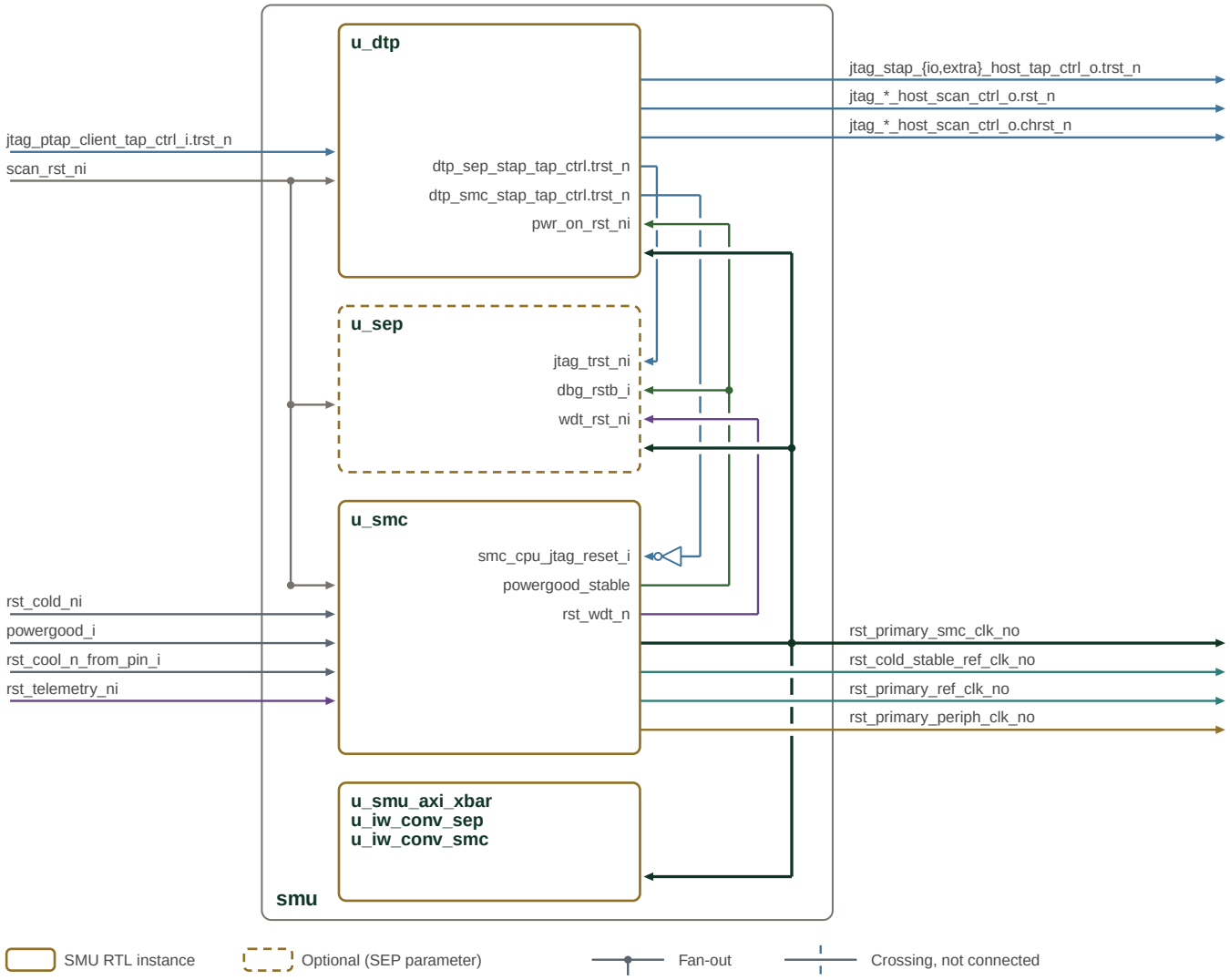


*Figure 7. SMU reset distribution*

## 2.1.4. Lifecycle and Feature Control

SEP lifecycle state and feature controls constrain the debug paths in DTP and the management functions in SMC, as shown in Figure 8.

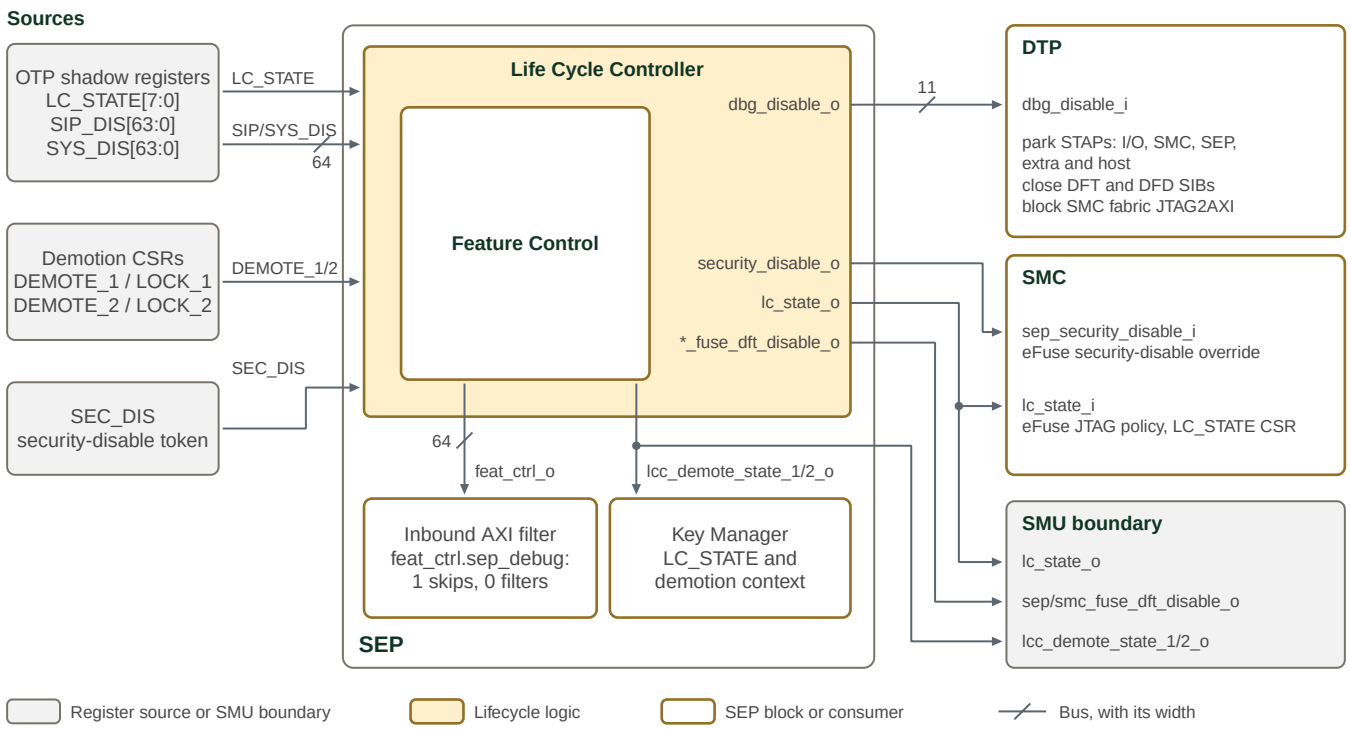


*Figure 8. SEP lifecycle and feature-control propagation to SMC and DTP*

In Figure 8, the OTP shadow registers, the demotion CSRs and the security-disable token feed the Life Cycle Controller, and each named output runs to the block that consumes it; a slashed line carries its bus width. Table 7 gives the effect of each control on its consumer. The demotion CSRs clear on SEP cold reset, and security disable overrides the normal feature restrictions.

*Table 7. SEP lifecycle and feature-control effects*

| Control | Consumer and effect |
|---|---|
| `dbg_disable_o` | DTP: disables selected STAPs, SIBs and SMC-fabric JTAG2AXI access |
| `lc_state_o` | SMC: eFuse JTAG access policy and the `LC_STATE` CSR. Also an SMU boundary output. |
| `security_disable_o` | SMC `sep_security_disable_i`: security-disable override for the eFuse wrapper |
| `sep_fuse_dft_disable_o`, `smc_fuse_dft_disable_o` | SMU boundary outputs for adopter DFT insertion; each gates its DFT OTP access paths |
| `feat_ctrl.sep_debug` | SEP inbound AXI filter: `1` bypasses the filter; `0` filters external traffic |
| `LC_STATE`, `lcc_demote_state_1/2` | Key Manager: lifecycle and demotion context. The demotion state also reaches the SMU boundary. |

The subsystem chapters (SEP, SMC, DTP) follow immediately; see the Composition section above for roles and cross-references.

## 2.1.5. Security Processor (SEP)

The Security Processor (SEP) is the chiplet Root of Trust. Its documentation is organized into focused topic pages so architecture, security policy, software interfaces, and generated register details can be read independently.

### 2.1.5.1. Overview

SEP combines a VeeR CPU, boot ROM, system peripherals, lifecycle controls, and cryptographic accelerators. The chapters below describe these functions in reading order; the block diagram shows their hardware connections.

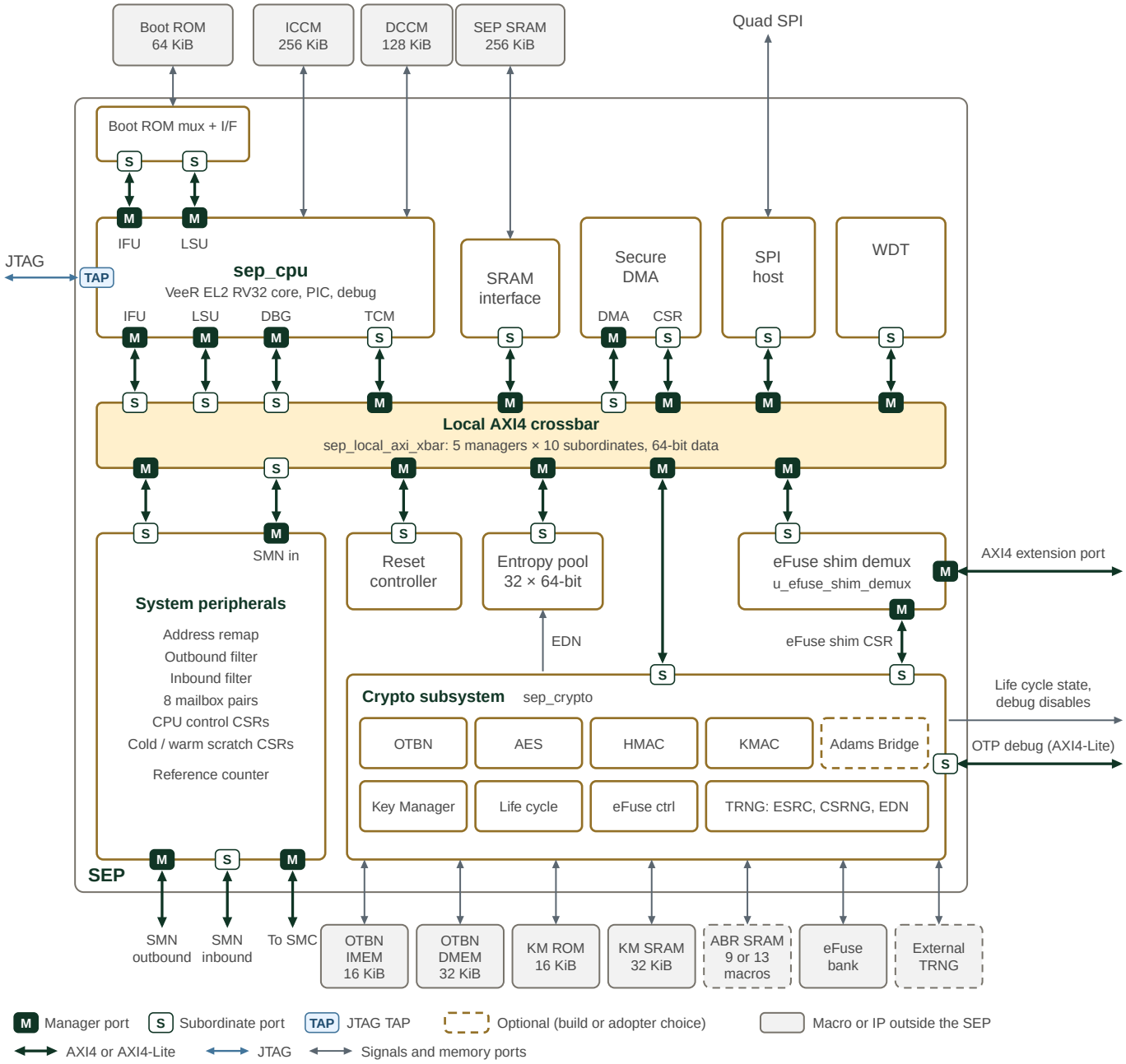


*Figure 9. SEP top-level microarchitecture*

Every AXI4 link carries a manager port (M) at the initiating end and a subordinate port (S) at the other. The local crossbar has five managers: the CPU IFU, LSU and debug ports, the DMA, and inbound SMN traffic after the system peripherals' inbound filter. Its ten subordinates are the CPU TCM port, the SRAM interface, the DMA registers, the SPI host, the watchdog, the system peripherals, the reset controller, the entropy pool, the crypto subsystem and the eFuse shim demux. SEP AXI4 connectivity gives the permitted paths. The IFU and LSU reach the boot ROM through a dedicated AXI mux rather than the crossbar, and the IFU crossbar port reaches only the SRAM. The eFuse shim demux sends accesses in the eFuse shim CSR window to the crypto subsystem and the rest of the external aperture to the AXI4 extension port.

The grey boxes are the memory macros and IP that the integration layer instantiates outside the SEP. The Adams Bridge and its SRAMs are present only when the build defines `SEP_ABR_EN`; the Adams Bridge uses nine SRAMs, plus four masked copies when `ABR_MASKING_EN` is set. The external TRNG is an adopter choice. Interrupt lines to the CPU PIC are not drawn; see Interrupts.

#### Responsibilities

SEP is the Root of Trust (RoT) of the chiplet with the highest security privilege. SEP provides a physically isolated execution environment for highly sensitive security services. The following are some examples:

- **Secure boot**: The SEP ROM BL0 image uses non-volatile public keys to conduct authentication verification on the loaded BL1 software images.
- **Secure lifecycle control**: SEP implements a secure lifecycle controller to manage function, debug and test access to the chiplet according to fuse configuration, in-field demotion, and secret token inputs.
- **Device attestation**: SEP offers support to device attestation by recording the boot process and device configuration followed attestation using public key infrastructure.
- **Cryptography service**: SEP offers secure key services to the rest of the system together with cryptography acceleration. SEP also implements True Random Number Generator (TRNG) to provide random numbers for SEP itself and the rest of the system.
- **User authentication**: Similar to secure boot, user authentication is another authentication service SEP offers. Besides feature control that can be tied to the authentication result, SEP software can also tie the authentication result to the secret key derivation to ensure that bypass of the user authentication leads to a useless system.

The SEP is responsible for ensuring that the chiplet and overall packaged system composed of multiple chiplets remains secure from adversaries. This means that SEP ensures the system

- Runs correctly with the intended firmware and data protected by digital signatures
- Guards against malicious data input to SEP, SMC and the debug/test infrastructure
- Prevents exfiltration of private or secret data on all exits, including side channels
- Prevents privilege escalation which would enable illegal operations

A consolidated description of OCAH use cases is available in Use Cases.

#### Threat Model

The threat model defines the attacks considered by SEP.

#### Attack Countermeasures

Attack Countermeasures describes the corresponding architectural, debug and test protections.

#### Clocks and Resets

SEP uses one system clock (`clk_i`) that clocks the CPU, interconnect and all cryptographic subsystems. Additional clock roots supply reference counter (`clk_ref_i`), watchdog timer (`clk_wdt_i`), entropy sampler (`entropy_rosc_sample_clk_i`) and CPU debug (`jtag_tck_i`). Clocks and Resets describes their distribution and links to software reset sequencing.

#### CPU

The CPU provides a JTAG debug interface, tightly coupled instruction and data memories, and two scratch register banks. The proposed Alarm Aggregation Processor is deferred to a later generation.

#### Boot ROM

The immutable BL0 firmware starts the SEP and authenticates the next boot stage. The Boot ROM chapter owns the hardware layout and boot-flow summary.

#### Interconnect

AXI4 connects the CPU, system peripherals and cryptographic subsystem. External interfaces include inbound and outbound system AXI paths, the peripheral extension and JTAG. UART and GPIO services can be reached through SMC.

#### Interrupts

The interrupt chapter identifies PIC sources and the separate NMI, timer, software-interrupt and memory-error paths.

#### Peripherals

DMA provides data movement and inline hashing. Mailboxes carry host service requests. SPI supplies flash access; the watchdog provides bark and bite events. The peripheral subsystem chapter describes their integration and extension interface.

#### Security and Lifecycle

The OTP fuse controller, token processing, security-disable logic, lifecycle controller and test-mode controls govern access to security assets and debug.

#### Cryptographic Subsystem

AES, HMAC-SHA2, KMAC-SHA3, OTBN and Adams Bridge provide acceleration. Entropy Source, DRBG and entropy distribution supply random material. Key Manager owns the Key and Policy Vault and accelerator key-sideload interfaces.

#### Memory Map

The memory map summarizes CPU and external address views and links to each block's register reference. Technology-specific ROM, SRAM, OTP and entropy macros sit outside SEP and connect through its dedicated interfaces. These include BL0 ROM, ICCM, DCCM, scratch SRAM, KM ROM/SRAM and OTBN IMEM/DMEM.

### 2.1.5.2. Threat Model

The SEP threat model considers remote software attacks, physical attacks and supply-chain attacks. These categories describe the threats considered; the explicit limits of protection are listed below.

Software attacks apply across deployments and are the primary attack vector for data center implementations. Physical attacks are localized and require close proximity; they are relevant to automotive and robotics applications. Supply-chain attacks involve changes to the design, production or provisioning of chips.

#### Software Attacks

- Malicious firmware modification during boot, overlay or application loading, or updates.
- Modification or exfiltration of private data, such as ML model weights and cryptographic keys.
- Forged attestation that mimics legitimate device behavior.

**Physical Attacks**

The model assumes that an attacker has full visibility of SRAM and OTP data state.

Physical threats considered include:

- Fault injection to bypass security checks, including clock and voltage glitching, laser and electromagnetic (EM) energy injection, and thermal gradients. The model also groups logically induced faults, such as Row Hammer DRAM attacks, in this category.
- Differential cryptanalysis for key recovery, and side-channel attacks to extract keys or other secret data.
- Hardware probing to observe or control data within a chiplet or in communication between chiplets.

**Supply-Chain Attacks**

- Insider modification of the design to disable or weaken security features or insert backdoors, and theft of hardware secrets such as security tokens.
- Modification of IC design masks to change behavior or insert trojans. For example, modified BL0 boot ROM contents could disable secure boot.
- Corruption of fuse provisioning to enable or disable features without authorization or leak device-unique secrets.

**Limits of Protection**

SEP does not explicitly protect against the following threats, even though some appear in the categories above:

- Hardware probing of communications between chiplets inside the package.
- Hardware probing of chiplets inside the package and certain physical attacks, such as bias fault injection.
- Thermal gradients used to influence the entropy source or induce timing faults.
- Supply-chain attacks.

### 2.1.5.3. Attack Countermeasures

Countermeasures include:

- Protection of BL0 boot ROM and subsequent secure boot via digital signature verification
- Glitch and error detection
  - SRAMs and other memory structures such as FIFOs and register files are protected by parity (byte-wise) or SECDED
- Temporal noise injection, masking, secret splitting and obfuscation on sensitive operations
- Transaction filtering and firewalling
- Functional and physical isolation of private data assets e.g. cryptographic keys
- Anti-replay facilities e.g. non-volatile counterj in OTP
- Obfuscate data stored in SRAM and OTP memories
- Hardware zero fill of memory by the secure DMA, with no software-prepared source buffer and an erase window that the engine checks on every write (Zero Fill)

### Debug and DFT Security Considerations

The debug facilities and DFT share the scan chain infrastructure for their respective tasks. This is a security hazard as certain secret values will be resident in the registers: tokens, secrets, cryptographic keys. Unless prevented, an attacker could halt the chip execution and extract the register state containing these secrets. This section describes the countermeasures we take to prevent this from happening.

One debug feature is scandump which provides access to all internal register states. We must ensure that a certain subset of registers holding secrets are never dumped out in this mode.

There are three scenarios of how registers are handled for DFT scan test and debug scandump:

1. Full DFT and Debug access
2. DFT access but no Debug access
3. No DFT and no Debug access

Consequently there will be two discrete scan chains to handle the registers in groups 1 and 2. The register instances (or module instances) will be labelled as belonging to one of the three groups and the DFT scan insertion tool will group the registers into the respective scan chains which will be managed for security in the TAP controller.

These three groups correspond to the SEP asset classes defined authoritatively in the Lifecycle Controller chapter (*Asset and Scan-Chain Protection*): group 1 = Class 3 (general), group 2 = Class 2 (crypto / KM secret-bearing, plus the fuse program path), group 3 = Class 1 (secrets and integrity-critical control fields).

### Debug and Scan Dump

Debug mode is enabled when in-field debug is granted, via demotion (DEMOTE_1 / DEMOTE_2) or an RMA debug state (RMA_SIP / RMA_CHIPLET). In this mode, the user has near full control and observability of the chip state, but we must enforce security restrictions.

**General logic (Class 3)** scan dump is gated by the `*_DBG` feature-control bits (sourced from the OTP DIS vectors; `1'b0` = enabled, `1'b1` = disabled, irreversible) and the LC state — a scope's general logic is dumpable only when its local gate (`SEP_DBG` / `CHIPLET_DBG`), the `SIP_DBG` debug-access path into the chiplet, and the LC state permit it.

**Secret-bearing logic (Class 2)** — crypto-engine key / round-key storage and the Key Manager key vault / shared key storage — is placed on a **dedicated secure scan chain** so DFT keeps coverage, but is **never scan-dumpable**, for any owner or LC state. The `_DBG bits decide whether that chain is reachable, as they do for any chain; what makes it undumpable is the command class. The DFT controller admits a *test` command on it only when `secure_tm = 1`, admits nothing at `secure_tm = 0`, and never admits a **dump** command at all. Under `secure_tm = 1` the secret sources (UID / `CLASS_KEY`) are disconnected from these consumers, so a permitted test shift moves no real secret.

**Secrets and integrity-critical fields (Class 1)** are excluded from every scan chain (never scannable):

- Class 1a (confidentiality): UIDs, `CLASS_KEY`, token-plaintext input registers (`*_TOKEN_I`), raw fuse read-back data.
- Class 1b (integrity): `LOCKS`/`LOCKS_SPARE`, `LC_STATE`, `SIP_DIS`, `SYS_DIS`, `*_TOKEN_DIGEST` shadow registers.

### Class 2 Asset Manifest and Naming

The authoritative class definitions, scan-chain policy, and gating rules are in the Lifecycle Controller chapter (*Asset and Scan-Chain Protection*); Key Manager specifics are in the Key Manager chapter (*Scan and Scan-Dump Protection*). This section records the implementation manifest. A complete module instance is classified as Class **2a** when it is the Key Manager, a device-secret-consuming cryptographic engine, an entropy engine, or sequential distribution logic that

retains key, seed, random, digest-working, or round-key state. Classifying complete instances deliberately includes their control flops and avoids fragile per-register marking.

The current SEP implementation has exactly 11 Class 2a RTL scan roots:

- `u_sep_crypto.u_hmac_wrapper_s3c_scan`
- `u_sep_crypto.u_sep_crypto_otbn_wrapper_s3c_scan`
- `u_sep_crypto.u_aes_wrapper_s3c_scan`
- `u_sep_crypto.u_kmac_wrapper_s3c_scan`
- `u_sep_crypto.u_sep_crypto_abr_wrapper_s3c_scan`
- `u_sep_crypto.u_sep_trng.u_entropy_source_s3c_scan`
- `u_sep_crypto.u_sep_trng.u_drbg_s3c_scan`
- `u_sep_crypto.u_axis_edn_crypto_s3c_scan`
- `u_sep_crypto.u_axis_edn_pool_s3c_scan`
- `u_sep_crypto.u_key_manager_s3c_scan`
- `u_trng_wrapper_s3c_scan`

The `_s3c_scan` suffix is reserved for Class 2 instance and generate-scope names. It is not a module-name or per-flop suffix. A nested `_n0_scan` name remains Class 1 and takes precedence over an enclosing `_s3c_scan` hierarchy.

Three writable companion SRAMs are also Class 2a assets: `u_km_sram`, `u_otbn_imem_sram`, and `u_otbn_dmem_sram`. Their macro instance names remain unchanged because `_s3c_scan` identifies RTL scan hierarchies; the owning memory-DFT/MBIST policy must provide the same never-dump outcome. `u_km_rom` is explicitly excluded because immutable executable contents cannot retain runtime secrets. The vendor TRNG core keeps its delivered DFT view beneath `u_trng_wrapper_s3c_scan`.

Class **2b** covers the fuse programming **control** and **status** MMRs of the SEP fuse controller. These hold no secret; they are never-dumpable because scan shifting drives arbitrary transient values through them, and a program-control register reached during an ordinary dump could burn fuses (see the Lifecycle Controller chapter, *Asset Classes*). Class 2b shares the secure chain and the command-class rule with Class 2a, so the `_s3c_scan` suffix applies to it as well. Unlike Class 2a it is **not** a whole-instance classification: the remainder of the fuse controller stays Class 3, so the implementation must identify the program-path scan roots and name them accordingly.

#### Secure Test Mode (SECURE_TM)

We assert a GPIO pin strap to enable secure test mode. This generates the SECURE_TM signal which is consumed by

- The DFT controller, which admits a **test** command on the Class 2 secure chain only while SECURE_TM is asserted. SECURE_TM does not enable scan test and authorizes nothing; it is the condition under which that chain will accept a test command at all.
  - Shadow registers of LOCKS/LOCKS_SPARE, LC_STATE, SIP_DIS, and SYS_DIS are non-scannable registers
  - Any downstream registers in the path from these registers to the feature control outputs or the fuse write-enable signals are also non-scannable, as are the SECURE_TM latch and the command-admission logic that consumes it
    - Dedicated test registers may be inserted in the path while in SECURE_TM mode to enable testing of downstream logic

- During scan test (`secure_tm = 1`) the secret-source outputs (UID / `CLASS_KEY`) are disconnected from the Class 2 consumers, so no real secret is shifted. On the Class 2 secure chain the DFT controller admits a test command only at `secure_tm = 1`, admits no command at `secure_tm = 0`, and never admits a dump command (never dumpable). See the Lifecycle Controller chapter, *`secure_tm` and the Secure Scan Chain*.
- Disable writing to any OTP fuse state to prevent updates.

- OTP controller
  - Disable *writing* of shadow registers: LOCKS/LOCKS_SPARE, LC_STATE, SIP_DIS and SYS_DIS
  - Other shadow register fields may optionally also have write disabled

#### 2.1.5.4. Clocks and Resets

**Clock Distribution**

The SEP system fabric, CPU and cryptographic subsystem receive `clk_i`. The CPU and some cryptographic blocks gate branches of this clock locally; these branches do not provide independently programmable CPU or accelerator frequencies. The other clock inputs serve reference counting, watchdog timing, entropy sampling and JTAG debug.

*Table 8. SEP clock inputs*

| Input | Use |
|---|---|
| `clk_i` | Main functional clock for the fabric, CPU, cryptographic subsystem and register interfaces. |
| `clk_ref_i` | Free-running reference clock counted by `REFERENCE_COUNTER`; the count crosses into `clk_i` for software access. |
| `clk_wdt_i` | Watchdog timer clock. The watchdog register interface uses `clk_i`, with clock-domain crossings to the timer core. |
| `entropy_rosc_sample_clk_i` | External entropy-sampling clock option. Each noise-source sampler selects between this input and the internal shared ring oscillator, then applies its sample-clock divider. |
| `jtag_tck_i` | CPU JTAG TAP clock; debug requests cross into `clk_i` through the debug-interface handshake. |

Figure 10 overlays these clock domains on the SEP top-level diagram. The CPU TAP runs on `jtag_tck_i`; local clock gating and derived clocks are described below. The CPU has a free-running branch and a pipeline branch gated according to halt/pause state, flush activity and clock override. SPI generates its serial clock locally. Entropy-source oscillators and sample clocks are distinct from the main functional clock. The diagram describes connectivity; operating frequencies and timing constraints belong to the platform integration.

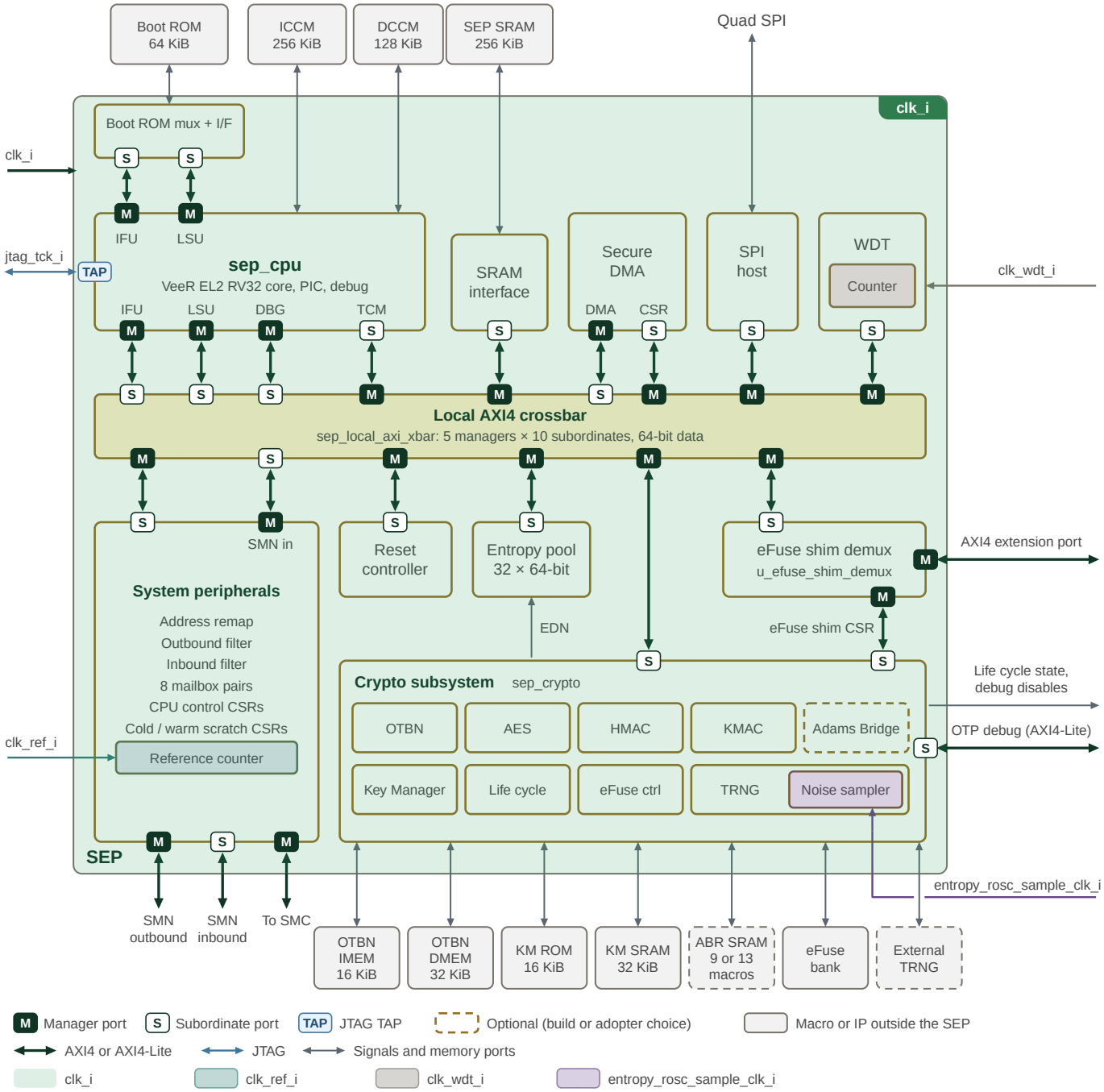


*Figure 10. SEP clock tree and clock domains*

### Reset Architecture

Reset scope determines which state survives a restart. In normal operation (with debug and scan overrides inactive), SEP uses the following reset boundaries.

*Table 9. SEP reset boundaries*

| Reset | Scope and release condition |
|---|---|
| Cold reset (`rst_ni`) | Resets the early infrastructure, including the local AXI crossbar, OTP controller, watchdog and reset-isolation sequencers. This infrastructure can operate before the boot-qualified SEP reset is released. |

| Reset | Scope and release condition |
|---|---|
| Boot-qualified SEP reset ( `sep_reset_n`) | The OTP controller qualifies release with fuse sensing complete, cold reset released and `ext_boot_seq_done_i` asserted. The SEP Reset Controller applies the SEP JTAG override and synchronizes release to `clk_i`. This reset feeds system peripherals and the reset-controller registers, and qualifies the software-controlled engine resets. |
| CPU reset (`sep_cpu_reset_n`) | Combines `sep_reset_n` with the aggregated SMC/SEP watchdog reset input, `wdt_rst_ni`. A watchdog reset restarts the CPU without asserting the boot-qualified SEP reset or the software-controlled cryptographic resets. It also clears the warm scratch register bank through the system CSR warm-reset input. |
| Software-controlled engine resets | Separate requests for Key Manager, OTBN, AES, HMAC, KMAC, Adams Bridge and the shared TRNG domain. Each sequencer isolates its required AXI paths and waits for the isolation acknowledgements before asserting reset. |

A watchdog restart therefore does not return the cryptographic engines or their software reset requests to cold-reset defaults; see the software reset section of the Programmer's Guide. The watchdog itself uses cold reset; its timer-domain reset release is synchronized to `clk_wdt_i`.

JTAG can override the SEP and individual engine resets. These overrides do not perform the software-request isolation sequence. Scan test enable selects `scan_rst_ni` at the SEP reset synchronizer and the final engine reset outputs. These debug/test paths are separate from normal software recovery.

The Platform Reset chapter describes the shared reset sources and subsystem effects. The SEP Reset Controller owns the software programming sequence, AXI isolation behavior and shared TRNG-domain details, including the distinction between the entropy subsystem and the Key Manager sampler.

Platform Reset

SEP Reset Controller

### SEP Reset Controller

The SEP Reset Controller provides software-controllable resets for the Key Manager, crypto accelerators, and internal TRNG complex. It is accessible via AXI4 at generated base address `0x1080_3000`.

Figure 11 shows the reset domains controlled by the SEP Reset Controller.

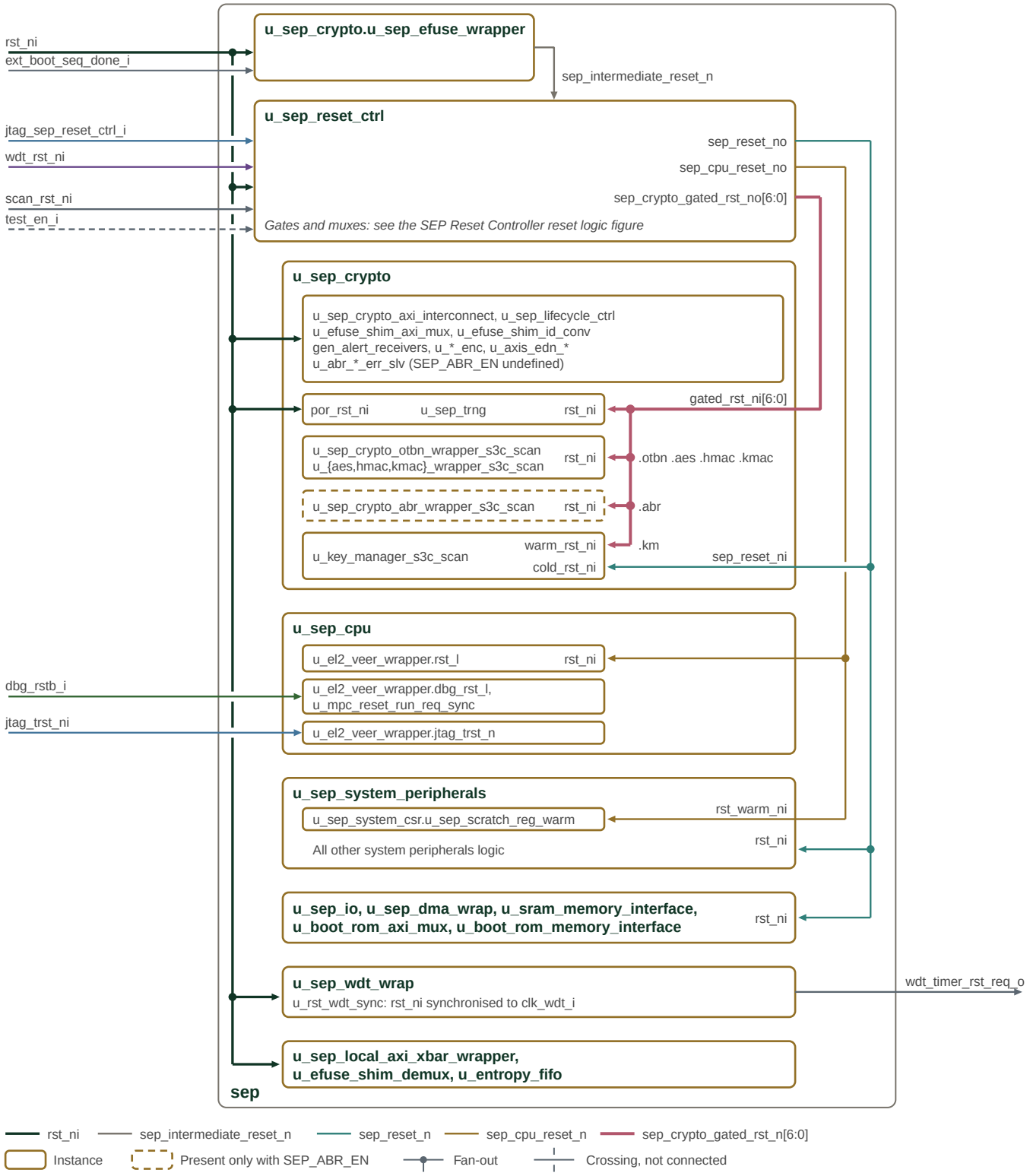


*Figure 11. SEP reset domains and reset sequencing*

Boundary reset inputs enter on the left and the generated resets run in the channel on the right, each net in its own colour, with the port it drives named at the instance edge. Inside each instance the figure shows which sub-instances each reset reaches. Figure 12 shows the logic inside `u_sep_efuse_wrapper` and `u_sep_reset_ctrl`, and Figure 32 shows the resets inside `u_sep_crypto`.

`u_sep_efuse_wrapper`, part of `u_sep_crypto`, ANDs `rst_ni`, `ext_boot_seq_done_i` and fuse-sense done and synchronizes the

result into sep_intermediate_reset_n. The Reset Controller builds its three outputs from it, as Figure 12 shows. The stacked row repeats once for each SW_RESET_N bit.

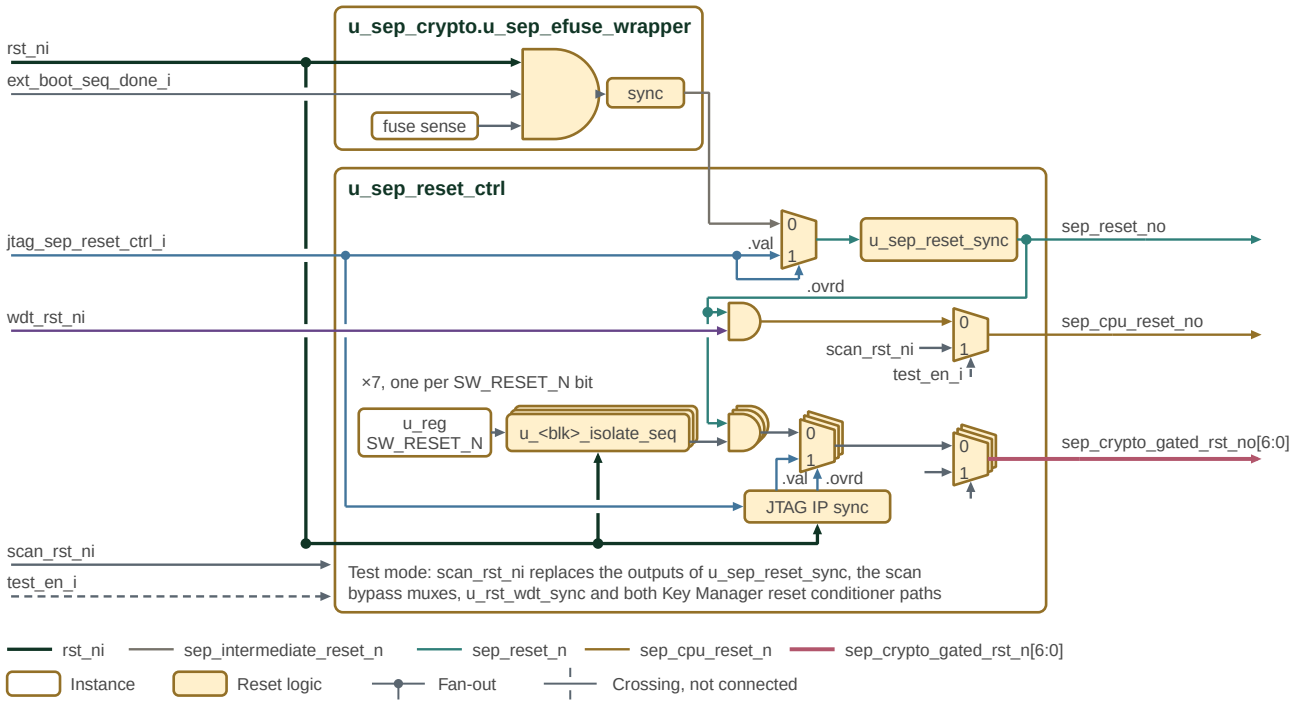


*Figure 12. SEP Reset Controller reset logic*

- sep_reset_no: the JTAG sep_reset_n override mux (u_sep_reset_ovrd_mux) selects sep_intermediate_reset_n or the JTAG value, and the 2-flop u_sep_reset_sync asserts the result asynchronously and releases it synchronously to clk_i. It resets the Reset Controller's own register block, the Key Manager cold domain, the system peripherals, the SPI host, the DMA and the SRAM and boot ROM interfaces.
- sep_cpu_reset_no: sep_reset_n ANDed with wdt_rst_ni (u_sep_cpu_rst_and), then the scan bypass u_sep_cpu_rst_scan_bypass. It resets the EL2 core (rst_l) and the warm scratch registers.
- sep_crypto_gated_rst_no[6:0]: for each bit, the output of u_<blk>_isolate_seq ANDed with sep_reset_n (u_<blk>_rst_and), then the per-IP JTAG override mux u_<blk>_rst_ovrd_mux, then the scan bypass u_<blk>_rst_scan_bypass. The per-IP override bits reach the muxes through the 2-flop synchronizers u_jtag_ip_ovrd_sync and u_jtag_ip_val_sync. The seven bits reset OTBN, AES, HMAC, KMAC, the TRNG, Adams Bridge and the Key Manager warm domain.

rst_ni resets the isolation sequencers and the JTAG override synchronizers. It also resets, directly, the crypto interconnect, life cycle controller, eFuse shim mux, alert receivers, encoders and EDN AXIS adapters, the TRNG power-on reset por_rst_ni, the watchdog, the local crossbar, the eFuse shim demux and the entropy pool. When the build does not define SEP_ABR_EN, the Adams Bridge error responders take rst_ni in place of the Adams Bridge wrapper. The watchdog synchronizes rst_ni into clk_wdt_i for its counter, and its bite leaves the SEP on wdt_timer_rst_req_o. dbg_rstb_i resets the EL2 debug module and u_mpc_reset_run_req_sync, and jtag_trst_ni resets the EL2 JTAG TAP. While test_en_i is high, scan_rst_ni replaces the outputs of u_sep_reset_sync, each scan bypass mux, u_rst_wdt_sync and both Key Manager reset conditioner paths.

### Software Reset Register

Table 10 defines the software reset controls and their reset values.

*Table 10. SEP SW_RESET_N fields*

| Bit | Field | Access | Default | Description |
|---|---|---|---|---|
| 0 | km_sw_rst_n | RW | 0 | KM reset (0=assert, 1=release) |
| 1 | otbn_sw_rst_n | RW | 1 | OTBN reset (0=assert, 1=release) |
| 2 | aes_sw_rst_n | RW | 1 | AES reset (0=assert, 1=release) |
| 3 | hmac_sw_rst_n | RW | 1 | HMAC reset (0=assert, 1=release) |
| 4 | kmac_sw_rst_n | RW | 1 | KMAC reset (0=assert, 1=release) |
| 5 | trng_sw_rst_n | RW | 1 | Shared entropy-source (ESRC), CSRNG, EDN and internal DRBG endpoint reset request (0=assert, 1=release); excludes the KM sampler |
| 6 | abr_sw_rst_n | RW | 1 | Adams Bridge reset (0=assert, 1=release) |

On power-up, `SW_RESET_N` resets to `0x7E`: KM is held in reset, while OTBN, AES, HMAC, KMAC, the TRNG, and Adams Bridge are released. The SEP host firmware writes `1` to release an IP from reset and writes `0` to request its software reset.

### Isolation and Reset Sequencing

Each `SW_RESET_N` bit drives a per-domain sequencer that isolates the domain's AXI paths before asserting its reset, so no transaction is in flight when reset asserts. Figure 13 shows every isolate stage in the crypto AXI interconnect and the sequencer that requests it.

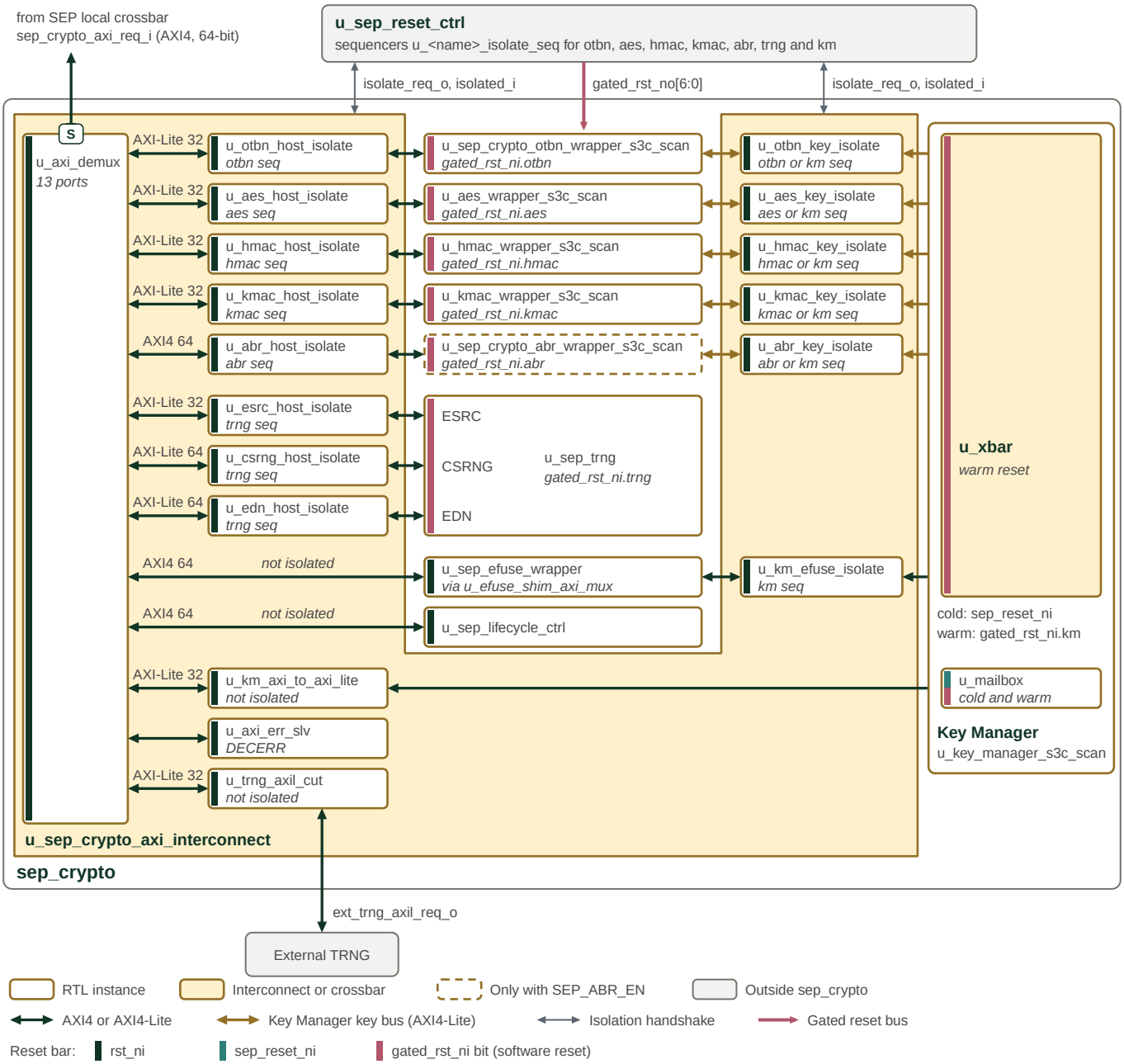


*Figure 13. SEP crypto AXI isolation*

Host paths leave `u_axi_demux` on the left, with the bus type after conversion above each arrow. Key Manager paths leave `u_xbar` on the right. Both kinds of isolate stage sit in `u_sep_crypto_axi_interconnect`, whose outline wraps around the engine wrappers it serves. The italic line in an isolate stage names the `u_sep_reset_ctrl` sequencer that requests it: `otbn seq` is `u_otbn_isolate_seq`, and `otbn or km seq` means that either sequencer requests it. The eFuse, lifecycle, Key Manager mailbox and external TRNG ports have no isolate stage. The coloured bar at the left of each instance shows its reset. The interconnect and all its isolate stages run on `rst_ni`, so they stay out of reset while a domain behind them is reset. Each engine wrapper, `u_sep_trng` and the Key Manager crossbar are reset by their `gated_rst_ni` bit, and the mailbox by both the Key Manager cold reset `sep_reset_ni` and its warm reset. Without `SEP_ABR_EN`, error responders that return DECERR replace the Adams Bridge wrapper behind both of its isolate stages.

A software reset proceeds as follows:

1. Firmware writes `0` to the domain's `SW_RESET_N` bit.

2. The sequencer requests isolation on the domain's paths. New transactions on them complete with SLVERR and read data `0x1501A7ED`.
3. Each isolate unit drains its accepted transactions, then reports isolated.
4. Once all of the domain's paths report isolated, the domain's reset asserts.
5. Firmware writes `1`. Reset and isolation are released together.

Table 11 lists the paths each domain isolates.

*Table 11. Paths isolated before each software reset*

| Domain | Paths isolated |
|---|---|
| OTBN, AES, HMAC, KMAC, Adams Bridge | Host path and Key Manager key path |
| Key Manager | All five key paths and the eFuse path |
| TRNG | ESRC, CSRNG, and EDN host paths |

A key path isolates when either endpoint is reset, so neither end can enter reset mid-transfer.

An OTBN, AES or KMAC software reset also cancels that engine's endpoints in the shared crypto EDN adapter, from the cycle its paths report isolated until its reset releases. EDN therefore keeps serving the engine while it drains. A word committed to the engine but not yet acknowledged is discarded rather than delivered to another client; other clients are unaffected.

The Key Manager mailbox, eFuse wrapper, lifecycle controller, and external TRNG ports are not isolated. None of these targets has its own software reset domain; the mailbox's SEP-facing side is in the Key Manager's cold domain. No `SW_RESET_N` request can therefore reset one mid-transaction.

The sequenced resets are ANDed with `sep_reset_n`, which holds every IP in reset until eFuse sensing completes during boot. JTAG can drive `sep_reset_n` in place of the eFuse reset through the SEP slice of the IC_RESET TDR (see IC_RESET Overrides in the DTP chapter), which releases it on a part whose fuse sensing does not complete. The per-IP JTAG override muxes are applied last, allowing debug to force reset without waiting for isolation.

#### Resetting the Key Manager

The sequenced Key Manager reset drives the Key Manager's warm reset, which restarts its CPU and volatile state but leaves its cold domain intact. That domain includes the SEP-facing side of the mailbox, so the SEP host can still reach the mailbox during the reset.

A soft reset requested through the Key Manager's `SOFT_RST_CODE` register takes the same warm path without isolation; the software reset section of the Programmer's Guide covers both requests. None is needed: the Key Manager CPU is the only master on its key and eFuse paths and completes each access before issuing the next, so no transaction is in flight when it writes `SOFT_RST_CODE`.

#### Resetting the TRNG

The TRNG reset bit controls one reset domain containing the entropy source, CSRNG, EDN, and DRBG endpoint state. ESRC, CSRNG, and EDN occupy three independent demux ports, so the TRNG sequencer isolates all three host paths and does not assert the TRNG reset until all three report isolated. Reset and isolation are released together.

The fabric entropy-pool AXI responder is not reset, so accesses remain responsive. While the TRNG reset is active, both post-mux EDN adapters and the pool synchronously clear their pending and buffered state, including the pool's half-packed word, FIFO contents, and fill-stall state.

Consumers are not part of the reset coordinator. Native EDN requests from AES, KMAC, and OTBN receive no acknowledgement while the post-mux adapter is clearing, remain asserted, and are serviced with fresh entropy after the chain restarts. Software may wait for those consumers to become idle and hold them in reset when policy requires aborting in-flight operations, but that is not required to assert the TRNG reset.

The Key Manager stream has no post-mux adapter, so TRNG reset does not scrub it. If the DRBG sampler is requesting or assembling a word when the reset asserts, the next DATA read can mix leftover bytes with new beats, or the sampler can flag a stream error. The SEP host must keep the Key Manager idle before asserting the TRNG reset. Holding KM in software reset is sufficient; hardware does not enforce the ordering.

The TRNG recovery sequence in the Programmer's Guide gives the firmware steps around this reset.

#### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### SEP Reset Control Address Map

Software resets for KM, OTBN, AES, HMAC, KMAC, the internal TRNG, and Adams Bridge.

*Table 12. SEP Reset Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | SW_RESET_N | RW | - |

#### SW_RESET_N

*Table 13. sep_reset_ctrl.SW_RESET_N fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:7 | `Reserved` | - | - | Reserved |
| 6 | `abr_sw_rst_n` | RW | 0x1 | Adams Bridge reset (0=assert, 1=release) |
| 5 | `trng_sw_rst_n` | RW | 0x1 | Internal TRNG reset request (0=assert, 1=release). Hardware drains and isolates the entropy-source, CSRNG, and EDN AXI ports before asserting the shared TRNG reset. |
| 4 | `kmac_sw_rst_n` | RW | 0x1 | KMAC reset (0=assert, 1=release) |
| 3 | `hmac_sw_rst_n` | RW | 0x1 | HMAC reset (0=assert, 1=release) |
| 2 | `aes_sw_rst_n` | RW | 0x1 | AES reset (0=assert, 1=release) |
| 1 | `otbn_sw_rst_n` | RW | 0x1 | OTBN reset (0=assert, 1=release) |
| 0 | `km_sw_rst_n` | RW | 0x0 | KM reset (0=assert, 1=release) |

### 2.1.5.5. CPU

SEP uses the 32-bit RISC-V VeeR EL2 CPU to run boot and security-service firmware. This chapter describes the SEP configuration, local memories, reset vector and debug integration. The EL2 Programmer's Reference Manual defines the core programming model. The Programmer's Guide gives the programming guidance for this block.

Figure 14 shows the CPU subsystem and its instruction, data, and external memory interfaces.

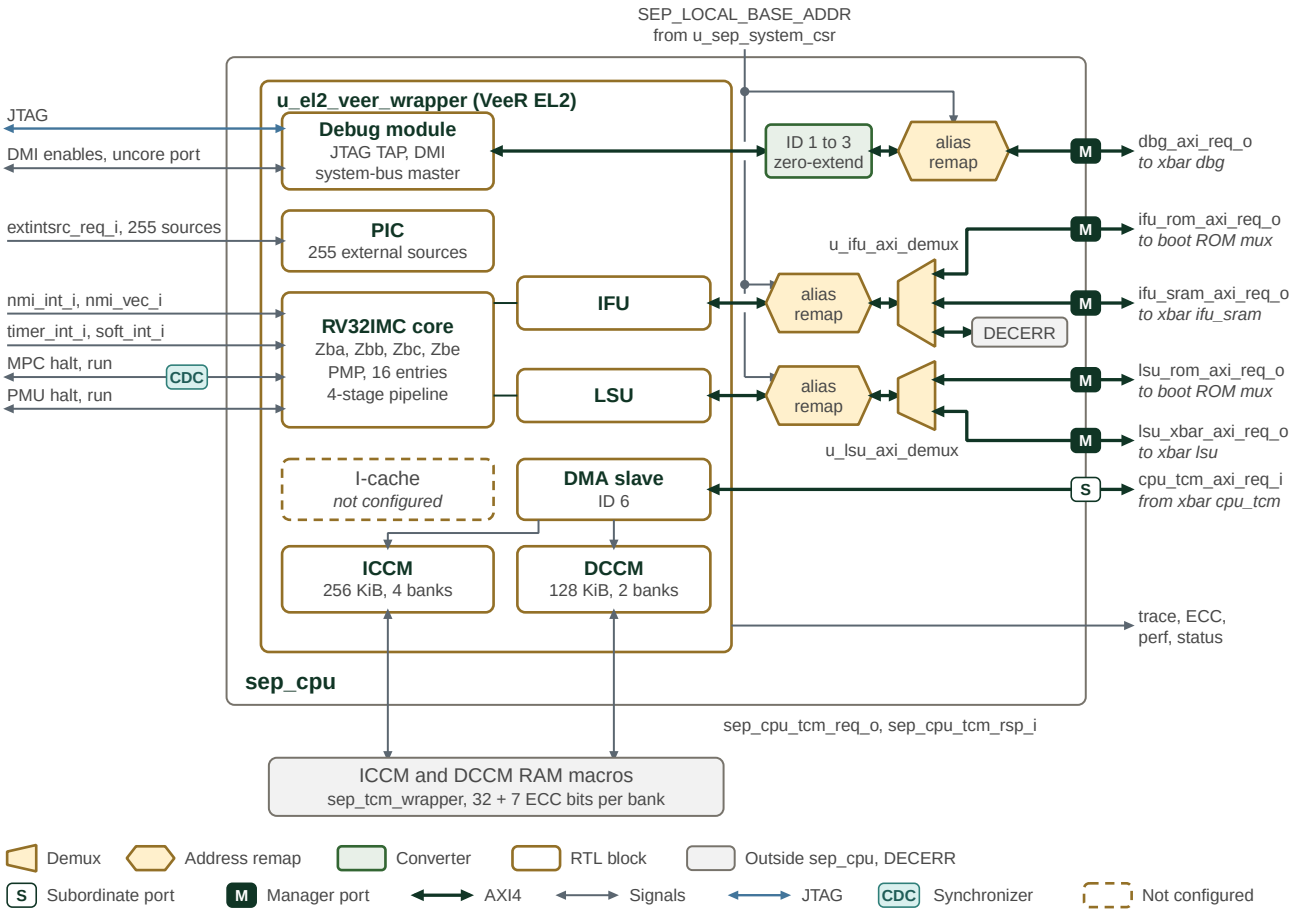


*Figure 14. SEP VeeR EL2 CPU subsystem and memory interfaces*

`sep_cpu` holds the VeeR EL2 core complex, `u_el2_veer_wrapper`, and the fabric elements on its bus masters; the figure uses the notation of the fabric figures. The ICCM and DCCM controllers are part of the core complex, but their RAM macros sit outside `sep_cpu` in `sep_tcm_wrapper`, each bank 32 data bits plus 7 ECC bits wide. The instruction cache is not configured. The DMA slave port is the local crossbar's `cpu_tcm` target, the backdoor through which the secure DMA preloads the ICCM and DCCM. The IFU, LSU and debug system-bus masters are AXI4 with 64-bit data and 3-bit IDs, and each carries `SepSourceId` in AxUSER. The debug master's 1-bit ID is zero-extended to 3 bits. Each master passes a local alias remap that rewrites the 768 MiB window at `SEP_LOCAL_BASE_ADDR` to `0x1000_0000`. The IFU demux then sends boot ROM addresses to the boot ROM mux, SEP SRAM addresses to the local crossbar and everything else to a DECERR responder, and the LSU demux sends boot ROM addresses to the boot ROM mux and everything else to the local crossbar. The core and its buses run on the SEP clock. `mpc_reset_run_req_i` passes a two-flop synchronizer in `sep_cpu`; the core synchronizes the NMI, timer, software interrupt and the other halt and run requests itself.

Every SEP bus master drives a fixed AxPROT, shown in Table 14. The value does not follow the hart's privilege mode, so a U-mode load still issues as privileged. The upstream VeeR EL2 port documentation lists `0b100` for IFU reads and `0b000` for the LSU and debug masters; the core drives the values in the table. The core ignores AxPROT on its DMA slave port.

*Table 14. AxPROT driven by the SEP bus masters*

| Master | AWPROT | ARPROT | Meaning |
|---|---|---|---|
| IFU, to the boot ROM and SEP SRAM | not issued | `0b101` | Privileged, secure, instruction |
| LSU, to the boot ROM and local crossbar | `0b001` | `0b001` | Privileged, secure, data |
| Debug system bus | `0b001` | `0b001` | Privileged, secure, data |

| Master | AWPROT | ARPROT | Meaning |
|---|---|---|---|
| Secure DMA master | `0b001` | `0b001` | Privileged, secure, data |

### Processor Core

The SEP build configures the core as follows. Optional VeeR capabilities must not be assumed to be present merely because the core supports them.

*Table 15. SEP processor configuration*

| Feature | SEP configuration |
|---|---|
| Instruction set | RV32IMC. The bit-manipulation build options enable Zba, Zbb, Zbc and Zbe; Zbf, Zbp, Zbr and Zbs are disabled. |
| Pipeline and privilege | Four-stage pipeline; user-mode support enabled. |
| Memory protection | 16 Physical Memory Protection (PMP) entries with Smepmp enabled. `pmpcfg` and `pmpaddr` CSRs control address ranges and access permissions. |
| Memory attributes | `mrac` independently selects cacheability and side-effect attributes for sixteen fixed 256 MiB regions. These attributes do not grant read, write or execute permission. |
| Interrupts and timers | Programmable Interrupt Controller (PIC), configured for 255 external sources, plus two internal timers that can operate independently or be linked. The timers use VeeR-specific `mip.MITIP0` and `mip.MITIP1` interrupts. |
| Debug | JTAG debug interface and an AXI4 debug master connected to the SEP fabric. |
| Dual Core Lock Step (DCLS) | Disabled in the checked-in SEP configuration. The integration exposes lockstep controls and status for builds that enable it; this does not establish functional-safety compliance. |

The Interrupts chapter owns the source-by-source PIC mapping: SEP CPU Interrupt Vector Map. The Boot ROM chapter describes secure boot and execution unlock.

### Local Memory

The configured closely coupled memories provide separate instruction and data storage. Their memory macros connect through the SEP CPU memory interface. The instruction cache is disabled in this configuration.

*Table 16. SEP closely coupled memories*

| Memory | Capacity | CPU base address | Protection |
|---|---|---|---|
| Instruction Closely Coupled Memory (ICCM) | 256 KiB | `0xC000_0000` | 7 ECC bits per 32-bit word |
| Data Closely Coupled Memory (DCCM) | 128 KiB | `0xC004_0000` | 7 ECC bits per 32-bit word |

### Dual Scratch Register Banks

The system CSR block supplies two software read/write scratch banks. Each contains eight 32-bit data fields in 64-bit register slots, giving an 8-byte stride. Both banks reset their data to zero. Hardware exposes the field values, but the current system CSR logic does not use them to control behavior.

*Table 17. Scratch-bank reset behavior*

| Bank | Reset and retention |
|---|---|
| Set 0, cold scratch | Cleared by `sep_reset_n`. Retains data through a watchdog-only CPU reset; retention is not guaranteed across reassertion of the boot-qualified SEP reset. |
| Set 1, warm scratch | Cleared by either `sep_reset_n` or `sep_cpu_reset_n`, so an aggregated SMC/SEP watchdog reset also clears this bank. |

The names distinguish watchdog-reset behavior; they do not mean that set 0 can only be cleared by a power-on event. The Clocks and Resets chapter defines these reset boundaries.

### Scratch Register Reference

### SEP Scratch Address Map

General-purpose SEP scratch register bank.

| NOTE | **Register Arrays:**<br>• **SCRATCH[8]**: 8 registers, base 0x0, stride 0x8 |
|---|---|

*Table 18. SEP Scratch Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x38 | SCRATCH[8] | RW | - |

### SCRATCH[8]

*Table 19. sep_scratch.SCRATCH[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `data` | RW | 0x0 | Scratch register for the CPU |

### Reset and Boot

The default reset vector points to BL0 in the Boot ROM. The debugger can change that vector as described below. Boot ROM owns the ROM layout, integrity limitations and BL0-to-BL1 boot sequence.

SEP Boot ROM.

### Configuration Straps

The platform boot-interface requirement is at least eight static configuration straps, preferably sixteen. This is a platform requirement, not a count of strap inputs on the CPU core. How the straps are sourced and captured is up to the adopter; firmware reads their values from the SMC strap registers (`STRAPS_LO` and `STRAPS_HI`).

BL0 uses platform configuration to select startup options, such as primary or secondary chiplet operation and normal or recovery boot. The board and boot firmware define the assignment of those options to pins; the configuration strap section of the Programmer's Guide describes reading them.

### Reset Vector

The SEP CPU reset vector is held in a test data register (TDR) in the EL2 debug TAP, selected by JTAG instruction opcode 0x18. The TDR resets to the Boot ROM base address on JTAG reset (TRST assertion or the Test-Logic-Reset TAP state). With that default vector, the CPU starts in the BL0 Root-of-Trust code.

An alternate address shifted into the TDR while the SEP CPU is held in reset redirects the first instruction fetch, which supports bring-up and failure analysis flows that execute from a memory other than the Boot ROM. The TDR stores address bits [31:1]; bit 0 is zero. The programmed value persists until it is rewritten or the TAP is reset; CPU reset alone

does not restore the default. The Programmer's Guide gives the redirect procedure.

### System Interfaces

#### DMA Interface

The DMA engine is described under Peripherals: Direct Memory Access (DMA)

#### Debug

The VeeR EL2 debug module is reached from the DTP SEP debug STAP through the core's JTAG TAP. TAP access is already gated by lifecycle state, so the DMI core aperture is left enabled at the SMU and does not take a second lifecycle check. The DMI uncore aperture is unused: the debug module's AXI master already reaches the rest of the SEP fabric.

#### Alarm Aggregation Processor (AGP)

The AGP is a proposed addition for a later SEP generation, aimed at edge applications such as automotive and robotics. It is not implemented in the first-generation SEP.

The proposed CPU-programmed monitor would collect and filter alarm events, detect suspected attacks and request countermeasures: wiping sensitive state or resetting selected components, subsystems or the SEP. These are future capabilities, not available alarm-response controls in the current integration.

### Memory Map and Register Reference

#### Memory Map

The SEP subsystem memory map gives the base addresses of the CPU-control and scratch banks. Offsets below are relative to the CPU-control block base. The detailed map describes the memory-mapped SEP CPU-control registers; RISC-V core CSRs such as `pmpcfg` and `mrac` are accessed by CSR instructions. The scratch-bank register layout appears under Local Memory above.

#### Detailed Register Map

#### SEP CPU Control Address Map

SEP CPU control and status registers: PKA and crypto configuration, boot and fast-memory enables, fuse-sense status, and interrupt controls.

*Table 20. SEP CPU Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x8 | CLOCK_GATE_CTRL | RW | TODO: clock-gate control - placeholder fields, not yet implemented. |
| 0x10 | REFERENCE_COUNTER | RW | - |
| 0x18 | TIMEOUT_INTERRUPT | R | Reserved - per-block timeout interrupt not yet implemented. |
| 0x20 | PKA_CTRL | RW | TODO: PKA control - placeholder fields, not yet implemented. |
| 0x28 | TIMEOUT_COUNT_DMA | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x30 | TIMEOUT_COUNT_SYS_IN | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x38 | TIMEOUT_COUNT_MAILBOX_INBOUND | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x40 | TIMEOUT_COUNT_MAILBOX_OUTBOUND | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x48 | TIMEOUT_COUNT_ENTROPY_WRITE | RW | Reserved - per-block timeout count/threshold not yet implemented. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x50 | TIMEOUT_COUNT_ENTROPY_READ | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x58 | TIMEOUT_COUNT_FILTER_OUT | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x60 | TIMEOUT_COUNT_ALIAS_REMAP | RW | Reserved - per-block timeout count/threshold not yet implemented. |
| 0x68 | TIMEOUT_ENABLE | RW | Reserved - per-block timeout enable not yet implemented. |
| 0x70 | TIMEOUT_CLEAR | W | Reserved - per-block timeout interrupt-clear not yet implemented. |
| 0x78 | TIMEOUT_MODE | W | Reserved - per-block timeout mode not yet implemented. |
| 0xB0 | SEP_TEST_CTRL | R | - |
| 0xC0 | SEP_GLOBAL_BASE_ADDR | RW | - |
| 0xC8 | SEP_LOCAL_BASE_ADDR | RW | - |
| 0xD0 | SEP_REGION_SIZE | RW | - |
| 0x100 | SMU_GLOBAL_BASE_ADDR | RW | - |
| 0x110 | SMU_REGION_SIZE | RW | - |
| 0x140 | SMC_FUSE_SENSE_STATUS | R | - |
| 0x150 | SEP_FUSE_SENSE_STATUS | R | - |
| 0x178 | SEP_SW_DEBUG | RW | - |
| 0x180 | SEP_NMI_VEC | RW | NMI vector address for VeeR-EL2 CPU. The default value is 0xC000_0100. The NMI interrupt is used for the WDT Bark interrupt.<br><br>**WARNING** the reset value is a placeholder, not a usable handler. Firmware must program this register with the address of its own NMI handler, and then set SEP_NMI_VEC_LOCK, before enabling any NMI source (the WDT bark in particular). If a bark arrives while the reset value is still in place, the CPU jumps to 0xC000_0100 in ICCM and executes whatever is there: arbitrary instructions if the loaded image covers that location, or an uncorrectable ICCM ECC error (which itself re-raises NMI) if it does not. |
| 0x188 | SEP_NMI_VEC_LOCK | RW | Lock the NMI vector address. Once locked, the NMI vector address cannot be changed. The NMI interrupt is used for the WDT Bark interrupt. |
| 0x190 | EXT_TRNG_SRC_SEL | RW | Per-stream external TRNG source selection for crypto subsystem entropy muxes. |
| 0x198 | EXT_TRNG_SRC_SEL_LOCK | RW | Lock the TRNG source selection. Once locked, EXT_TRNG_SRC_SEL cannot be changed until reset. |
| 0x1A0 | KM_WIPE_CTRL | RW | Key Manager emergency wipe control. |
| 0x1A8 | DMA_BUS_ERR_STATUS | R | Secure DMA bridge fault status. Each bit is latched in hardware and held until the corresponding DMA_BUS_ERR_CLEAR write. These bits also drive dedicated PIC sources.<br><br>Neither bit is cleared by a CPU-only reset (the WDT resets the CPU without resetting the DMA), so a bit read as set immediately after boot may describe a fault from before that reset rather than a new one. |
| 0x1B0 | DMA_BUS_ERR_CLEAR | W | Clear the DMA_BUS_ERR_STATUS bits. Writing 1 emits a single-cycle clear pulse to both latches. A fault arriving in the same cycle as the clear still latches, so acknowledging cannot drop a concurrent fault. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1B8 | PERIPH_BUS_ERR_STATUS | R | Peripheral register-bridge fault status, one bit per block. Each bit latches when a CPU access to that block's registers returned a TL-UL error response (unmapped or misaligned offset), and is held until the matching PERIPH_BUS_ERR_CLEAR bit is written. The faulting access also received SLVERR synchronously.<br><br>All bits OR together into a single PIC source, so read this register to identify which block faulted.<br><br>No bit is cleared by a CPU-only reset (the WDT resets the CPU without resetting the peripherals), so a bit read as set immediately after boot may describe a fault from before that reset rather than a new one. |
| 0x1C0 | PERIPH_BUS_ERR_CLEAR | W | Clear PERIPH_BUS_ERR_STATUS bits. Writing 1 to a bit emits a single-cycle clear pulse to that block's latch; bit order matches PERIPH_BUS_ERR_STATUS. Clears are per-block so acknowledging one block cannot discard a fault another block raised between the status read and this write. A fault arriving in the same cycle as its own clear still latches. |
| 0x1000 | SEP_VERSION_ID | R | - |

### CLOCK_GATE_CTRL

*Table 21. sep_cpu_ctrl.CLOCK_GATE_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `pka_cg_enable` | RW | 0x0 | Reserved - PKA clock-gate enable, not yet implemented. |

### REFERENCE_COUNTER

*Table 22. sep_cpu_ctrl.REFERENCE_COUNTER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rc` | RW | 0x0 | SEP REFERENCE COUNTER, counting after deassert of cold reset. Write once, then poll this register until the readback reflects the written value before writing again. The counter keeps advancing, so the readback is at or just above the written value rather than equal to it. |

### TIMEOUT_INTERRUPT

*Table 23. sep_cpu_ctrl.TIMEOUT_INTERRUPT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | R | 0x0 | Reserved - timeout interrupt status not yet implemented. |

### PKA_CTRL

*Table 24. sep_cpu_ctrl.PKA_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `pka_noise_src_valid` | RW | 0x0 | Reserved - PKA noise source valid, not yet implemented. |
| 1 | `pka_noise_src` | RW | 0x0 | Reserved - PKA noise source select, not yet implemented. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `pka_dpa_disable` | RW | 0x0 | Reserved - disable DPA countermeasures in PKA, not yet implemented. |

### TIMEOUT_COUNT_DMA

*Table 25. sep_cpu_ctrl.TIMEOUT_COUNT_DMA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_SYS_IN

*Table 26. sep_cpu_ctrl.TIMEOUT_COUNT_SYS_IN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_MAILBOX_INBOUND

*Table 27. sep_cpu_ctrl.TIMEOUT_COUNT_MAILBOX_INBOUND fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_MAILBOX_OUTBOUND

*Table 28. sep_cpu_ctrl.TIMEOUT_COUNT_MAILBOX_OUTBOUND fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_ENTROPY_WRITE

*Table 29. sep_cpu_ctrl.TIMEOUT_COUNT_ENTROPY_WRITE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_ENTROPY_READ

*Table 30. sep_cpu_ctrl.TIMEOUT_COUNT_ENTROPY_READ fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_FILTER_OUT

*Table 31. sep_cpu_ctrl.TIMEOUT_COUNT_FILTER_OUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_COUNT_ALIAS_REMAP

*Table 32. sep_cpu_ctrl.TIMEOUT_COUNT_ALIAS_REMAP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout count not yet implemented. |

### TIMEOUT_ENABLE

*Table 33. sep_cpu_ctrl.TIMEOUT_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | RW | 0x0 | Reserved - timeout monitors not yet implemented. |

### TIMEOUT_CLEAR

*Table 34. sep_cpu_ctrl.TIMEOUT_CLEAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | W 1P | 0x0 | Reserved - timeout monitors not yet implemented. |

### TIMEOUT_MODE

*Table 35. sep_cpu_ctrl.TIMEOUT_MODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `reserved` | W | 0x0 | Reserved - timeout monitors not yet implemented. |

### SEP_TEST_CTRL

*Table 36. sep_cpu_ctrl.SEP_TEST_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `fast_spi_en` | R | 0x0 | Fast SPI Enable |
| 30 | `fast_iccm_en` | R | 0x0 | Fast ICCM Enable |
| 29 | `fast_dccm_en` | R | 0x0 | Fast DCCM Enable |
| 28 | `fast_sram_en` | R | 0x0 | Fast SRAM Enable |
| 27 | `fast_pka_en` | R | 0x0 | Fast PKA Enable |
| 26 | `sep_standalone` | R | 0x0 | SEP is standalone (not part of eg SMU) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 25:0 | `Reserved` | - | - | Reserved |

### SEP_GLOBAL_BASE_ADDR

*Table 37. sep_cpu_ctrl.SEP_GLOBAL_BASE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `addr` | RW | 0x0 | - |

### SEP_LOCAL_BASE_ADDR

*Table 38. sep_cpu_ctrl.SEP_LOCAL_BASE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `addr` | RW | 0xD0000000 | - |

### SEP_REGION_SIZE

*Table 39. sep_cpu_ctrl.SEP_REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `size` | RW | 0x1000000 | - |

### SMU_GLOBAL_BASE_ADDR

*Table 40. sep_cpu_ctrl.SMU_GLOBAL_BASE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `addr` | RW | 0x80000000 | - |

### SMU_REGION_SIZE

*Table 41. sep_cpu_ctrl.SMU_REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `size` | RW | 0x40000000 | - |

### SMC_FUSE_SENSE_STATUS

*Table 42. sep_cpu_ctrl.SMC_FUSE_SENSE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `smc_fuse_sense_done` | R | 0x0 | This bit indicates when SMC's fuse sense is completed. While fuse sense if ongoing, SMC fabric will be in reset. Any requests to the SMC will hang. When this is 1, it is safe to send requests to the SMC. |

### SEP_FUSE_SENSE_STATUS

*Table 43. sep_cpu_ctrl.SEP_FUSE_SENSE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `sep_fuse_sense_done` | R | 0x0 | This reg is used to indicate the fuse sense status of SEP. |

### SEP_SW_DEBUG

*Table 44. sep_cpu_ctrl.SEP_SW_DEBUG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `sep_sw_debug` | RW | 0x0 | software writtable debug information. This reg is routed to CLA |

### SEP_NMI_VEC

*Table 45. sep_cpu_ctrl.SEP_NMI_VEC fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:1 | `nmi_vec` | RW | 0x60000080 | NMI vector address [31:1] for VeeR-EL2 CPU. Bit 0 is always 0 (aligned). Address where CPU jumps on Non-Maskable Interrupt. This is used for the WDT Bark interrupt. The reset value does not point at a valid handler — see the register description for the required programming order. |
| 0 | `rsvd` | R | 0x0 | Reserved, bit 0 is always 0 (aligned) |

### SEP_NMI_VEC_LOCK

*Table 46. sep_cpu_ctrl.SEP_NMI_VEC_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `lock` | RW1S | 0x0 | Lock the NMI vector address. Once locked, the NMI vector address cannot be changed. The NMI interrupt is used for the WDT Bark interrupt. |

### EXT_TRNG_SRC_SEL

*Table 47. sep_cpu_ctrl.EXT_TRNG_SRC_SEL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2:0 | `sel` | RW | 0x7 | TRNG source select per stream. Bit N selects the source for stream N (bit0 = Key Manager, bit1 = crypto blocks, bit2 = entropy pool). 0 = internal DRBG output, 1 = external TRNG (default). |

### EXT_TRNG_SRC_SEL_LOCK

*Table 48. sep_cpu_ctrl.EXT_TRNG_SRC_SEL_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `lock` | RW1S | 0x0 | Lock TRNG source selection. Once set, cannot be cleared by software. |

### KM_WIPE_CTRL

*Table 49. sep_cpu_ctrl.KM_WIPE_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `wipe_state` | RW | 0x0 | Key Manager wipe state. A rising edge zeroes all KPV entries and sets the WIPE_STATE interrupt status. |

### DMA_BUS_ERR_STATUS

*Table 50. sep_cpu_ctrl.DMA_BUS_ERR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `host_path_err` | R | 0x0 | DMA-master-path fault: either the fabric returned a non-OKAY AXI response to a DMA transfer, or a TL-UL command-integrity check failed on a command issued by the DMA. The latter is a fault-detection event with no other reporting path. |
| 0 | `reg_path_err` | R | 0x0 | Register-path bus error: a CPU access to a Secure DMA register returned a TL-UL error response (unmapped or misaligned offset). The faulting access also received SLVERR synchronously. |

### DMA_BUS_ERR_CLEAR

*Table 51. sep_cpu_ctrl.DMA_BUS_ERR_CLEAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `clr` | W 1P | 0x0 | Write 1 to clear both DMA_BUS_ERR_STATUS bits. Always reads 0. |

### PERIPH_BUS_ERR_STATUS

*Table 52. sep_cpu_ctrl.PERIPH_BUS_ERR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:7 | `Reserved` | - | - | Reserved |
| 6 | `wdt` | R | 0x0 | Watchdog timer register bridge error. |
| 5 | `edn` | R | 0x0 | EDN register bridge error (DRBG EDN path). |
| 4 | `csrng` | R | 0x0 | CSRNG register bridge error (DRBG CSRNG path). |
| 3 | `otbn` | R | 0x0 | OTBN register bridge error. |
| 2 | `kmac` | R | 0x0 | KMAC register bridge error. |
| 1 | `hmac` | R | 0x0 | HMAC register bridge error. |
| 0 | `aes` | R | 0x0 | AES register bridge error. |

#### PERIPH_BUS_ERR_CLEAR

*Table 53. sep_cpu_ctrl.PERIPH_BUS_ERR_CLEAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:7 | `Reserved` | - | - | Reserved |
| 6 | `wdt` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.wdt. Always reads 0. |
| 5 | `edn` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.edn. Always reads 0. |
| 4 | `csrng` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.csrng. Always reads 0. |
| 3 | `otbn` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.otbn. Always reads 0. |
| 2 | `kmac` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.kmac. Always reads 0. |
| 1 | `hmac` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.hmac. Always reads 0. |
| 0 | `aes` | W 1P | 0x0 | Write 1 to clear PERIPH_BUS_ERR_STATUS.aes. Always reads 0. |

#### SEP_VERSION_ID

*Table 54. sep_cpu_ctrl.SEP_VERSION_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `version_id` | R | 0xDEADBEEF | Version ID for the CPU |

### 2.1.5.6. Boot ROM

The SEP maps a 64 KiB immutable Boot ROM aperture at `0x1004_0000` through `0x1004_FFFF`. The SEP CPU wrapper sets the VeeR EL2 `RESET_VEC` parameter to the base of this region, so a cold boot begins in the BL0 Root-of-Trust firmware.

#### ROM Architecture

The CPU instruction-fetch and load/store interfaces reach the Boot ROM over a dedicated path outside the SEP local AXI crossbar. The SEP ROM interface shim converts the CPU memory request into the request, 64-bit word address, and read-data signals used by the ROM macro. Stores receive a response but are not forwarded to the macro and do not modify the ROM.

The open integration instantiates a 16K-by-64-bit generic ROM, a physical 128 KiB array. The generated SEP address map and production linker expose only the first 64 KiB. The unused physical capacity is not part of the BL0 software address contract.

The ROM stores executable code, immutable metadata, read-only data, and the initial image of writable data; writable data, the stack and the BL0-to-BL1 state run from the 128 KiB DCCM at `0xC004_0000`. The memory and security section of the SEP Boot ROM appendix in the Programmer's Guide gives the linker layout.

BL0 stages the selected manifest and payload in SEP SRAM beginning at `0x1000_0000`. A validated SEP BL1 image is copied by the secure DMA to the 256 KiB ICCM at `0xC000_0000`, where the instruction-fetch unit executes it. The CPU load/store unit cannot access ICCM, so BL0 cannot populate it with an ordinary software copy.

#### Hardware Configuration and Integrity

A future ROM patching facility using OTP backup storage is planned for the second generation; it is not implemented in the first generation.

The open generic ROM model has a 64-bit data path and no ECC or integrity-error sideband. Technology-specific integrity behavior is therefore outside this integration's visible hardware contract.

#### Boot Flow

The production BL0 firmware selects SPI flash or an SMC-staged image from the captured boot straps. It validates the manifest, applies the lifecycle and secure-boot policy, checks the payload, locks sensitive fuse reads, and transfers control to BL1. The SEP Boot ROM appendix of the Programmer's Guide describes that behavior and its build artifacts in detail.

#### BL0 Secure Boot and BL1 Execution Unlock

BL0 brings the SEP into a runnable state after power-on reset, validates a boot manifest and its payload, unlocks BL1 execution in the CPU PMP and jumps to it. Any failed check ends the boot: a failure the backup manifest slot recovers from is a warning, and exhausting every slot leaves the processor in a fail loop until reset. The ordered steps (CPU and PMP setup, fuse and lifecycle evaluation, SMC synchronization, boot-source selection, manifest and payload validation, secure boot, demotion, attestation and hand-off), their requirement identifiers and the implementation-status matrix are in the SEP Boot ROM appendix of the Programmer's Guide.

#### Firmware Reference

The production ROM firmware is documented in the SEP Boot ROM appendix of the Programmer's Guide.

### 2.1.5.7. Interconnect

SEP uses a hierarchical AXI4 crossbar fabric. Masters initiate transactions; slaves connect directly or through bridges for AXI4-Lite and OpenTitan TL-UL interfaces. The `axi_to_axi_lite` bridge splits bursts into individual transactions and retains the AXI IDs needed for responses; register regions accept single beats only (Single-Beat Register Access). Write strobes remain available on AXI4-Lite; `axi_lite_to_tlul` translates them into TL-UL byte masks. Each peripheral's register definition determines which writes are permitted.

#### Topology

#### Top Level AXI4

Figure 15 shows the crossbar connectivity implemented in `hw/sys/sep/rtl/crossbars/sep_local_axi_xbar.sv`, along with the dedicated Boot ROM path that lies outside the crossbar. The crossbar has five initiators (CPU IFU, CPU LSU, CPU DBG, DMA, and the System Interface) and ten targets. The CPU masters and the DMA issue single-beat transactions only; the System Interface may issue bursts.

A few access paths are intentionally outside this crossbar:

- **Boot ROM** is accessed by the CPU IFU and LSU through a dedicated point-to-point path, not through this crossbar.
- The CPU's own **ICCM/DCCM** are tightly coupled to the core and are not reached through this crossbar by the IFU/LSU. The **CPU TCM** target below is a backdoor AXI path into the ICCM/DCCM, used by the DMA engine to preload code/data.
- The *Alarm AGP is a feature deferred to the next generation* and is not present.

The **System Periph** target aggregates the System Bus interface, the dual scratch register bank, the AP/STEE remap regions, and the outbound routing to SMC/SMN resources.

The **Entropy Pool** target is `sep_entropy_fifo` (0x1095_0000): a 64-bit read-only entropy drain aperture, filled

autonomously by a native EDN endpoint on a separate hardware path (not this crossbar). Writes to it terminate with `BRESP=SLVERR`.

The **SRAM** target serves INCR bursts of every length and single-beat bursts of every type. A multi-beat FIXED or WRAP burst is refused before it reaches the memory: the write terminates with `BRESP=SLVERR` and leaves SRAM unchanged, and every beat of the read returns `RRESP=SLVERR`.

The **System Interface** initiator is the external inbound master. It does not attach to this crossbar directly; it enters SEP through the dedicated `smn_inbound` port and the inbound traffic filter (block-by-default), then through the `sep_system_peripherals` inner crossbar. That inner crossbar reaches the **mailboxes** and **system CSRs** directly, and loops everything else back into this crossbar's System Interface initiator. The inbound filter controls which addresses the external master may reach; it is block-by-default after POR and configured by SEP firmware.

*Table 55. SEP AXI4 connectivity terms*

| Label | Meaning |
|---|---|
| CPU IFU | CPU instruction-fetch unit; initiates instruction reads. |
| CPU LSU | CPU load/store unit; initiates data reads and writes. |
| CPU debug (`dbg`) | CPU debug access path. |
| DMA | Direct memory access engine; moves data without CPU load/store instructions. |
| CPU TCM | Tightly coupled memory backdoor: DMA access to the instruction closely coupled memory (ICCM) and data closely coupled memory (DCCM). |
| SRAM (`sram`) | Static random-access memory used for SEP working storage. |
| DMA registers (`dma_csr`) | DMA control and status registers. |
| Watchdog (`sep_wdt`) | Watchdog timer. |
| Reset controller (`sep_reset_ctrl`) | SEP software reset controller. |
| Crypto (`sep_crypto`) | Cryptographic accelerators (AES, HMAC, KMAC, OTBN), entropy-source registers, eFuse controller, lifecycle controller, Key Manager mailbox, and external TRNG window. |
| System peripherals (`sep_system_peripherals`) | System-peripheral routing, remapping, mailboxes and system registers. |
| SPI host (`sep_io`) | Local input/output peripheral path, including SPI boot-flash access. |
| Entropy pool (`entropy_fifo`) | Read-only aperture for entropy supplied by the entropy distribution network (EDN). |
| Extension, eFuse shim (`sep_external`) | Outbound AXI4 port for resources outside the local crossbar; the eFuse shim CSR window is diverted by `u_efuse_shim_demux` to the eFuse wrapper. |
| SMN inbound (`ext`) | Filtered inbound path from the System Management Network (SMN). |
| Boot ROM | Read-only boot memory on the dedicated CPU fetch/load path. |

| Crossbar targets | Initiators | | | | |
|---|---|---|---|---|---|
| | CPU IFU ifu_sram | CPU LSU lsu | CPU debug dbg | DMA dma | SMN inbound ext |
| cpu_tcm (CPU TCM) | — | — | — | ● | — |
| sram (SEP SRAM) | ● | ● | ● | ● | ● |
| dma_csr (DMA registers) | — | ● | ● | — | ● |
| sep_wdt (watchdog) | — | ● | ● | ● | ● |
| sep_reset_ctrl (reset controller) | — | ● | ● | ● | — |
| sep_crypto (crypto) | — | ● | ● | ● | ● |
| sep_system_peripherals (system peripherals) | — | ● | ● | ● | — |
| sep_io (SPI host) | — | ● | ● | ● | ● |
| entropy_fifo (entropy pool) | — | ● | ● | — | ● |
| sep_external (extension, eFuse shim) | — | ● | ● | ● | ● |
| **Dedicated path, outside the crossbar** | | | | | |
| boot ROM (u_boot_rom_axi_mux) | ● | ● | — | — | — |

● Connected — Not connected

*Figure 15. SEP AXI4 connectivity. Filled circles indicate connections; dashes indicate no connection.*

### Fabric Topology

The input fabric admits external traffic through the inbound filter and system-peripheral decode. The local fabric connects SEP initiators to local resources, while the output fabric selects remapped or direct outbound paths.

In the fabric diagrams, labels such as `ID 3 to 4` give the input and output AXI transaction-ID widths in bits; they do not count ports or data bits. The wider side of the symbol indicates the side with multiple ports. Bus labels identify address and ID widths where they change. The filter wrapper checks reads and writes independently, forwards permitted requests, and routes rejected requests to an AXI error responder.

The input paths are shown in Figure 16, the local resource connections in Figure 17, and the outbound paths in Figure 18. The three figures share one notation. Crossbars are rectangles, muxes and demuxes are trapezoids with the wide side towards the many ports, address remaps are hexagons and traffic filters are octagons. Green boxes are converters: address-width, data-width, ID-width and protocol conversion, and register cuts. Thick lines are AXI4 and thin lines AXI-Lite or TL-UL; every AXI data path is 64 bits wide unless a converter narrows it. The labels give the address and ID widths where they change, and the port badges mark where a path crosses the module boundary.

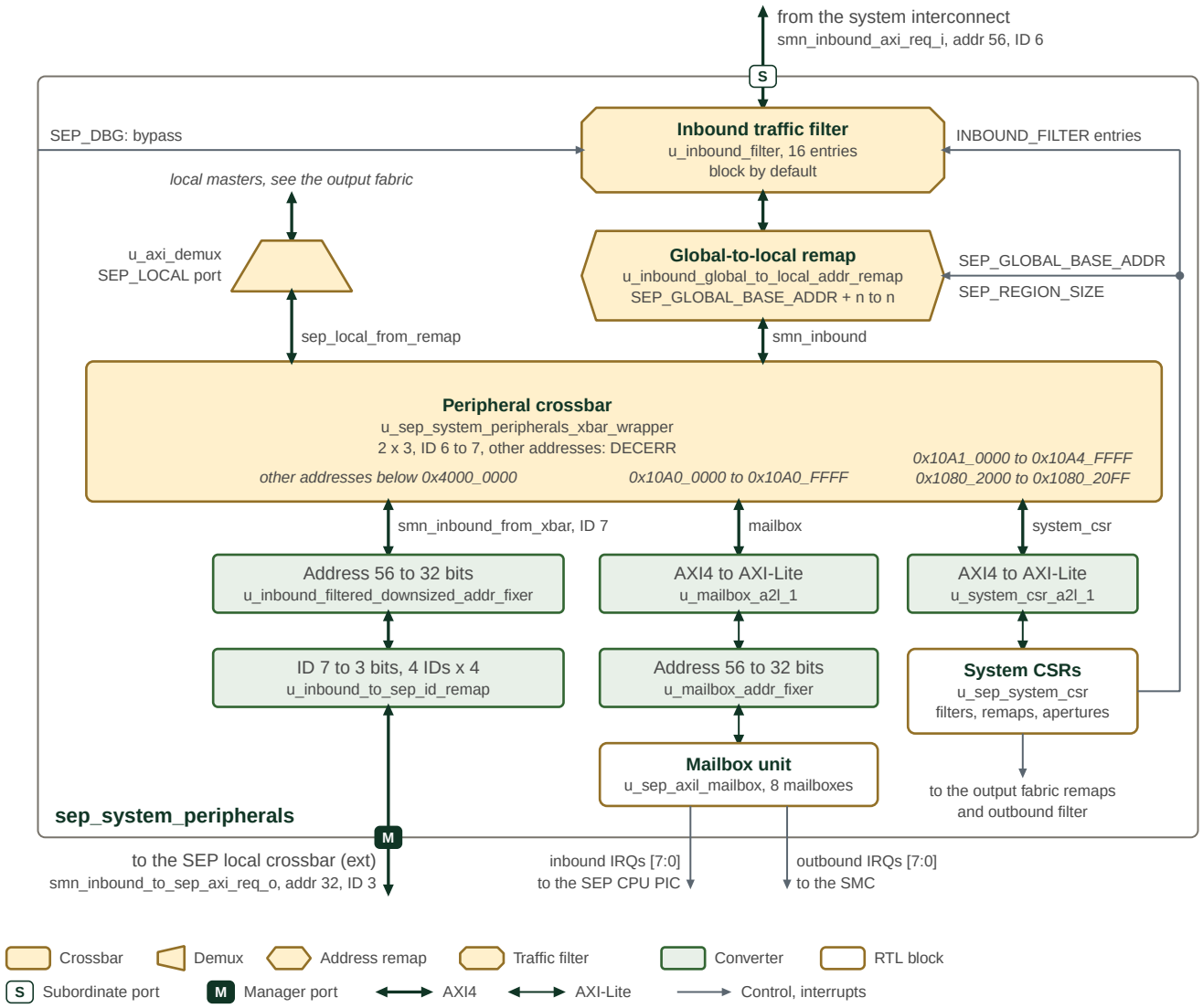


*Figure 16. SEP input fabric*

Figure 16 follows a request from the system interconnect through `sep_system_peripherals`. The inbound filter checks it first; while SEP-scope debug is enabled, `SEP_DBG` bypasses the filter. The global-to-local remap then rebases the `SEP_REGION_SIZE` window at `SEP_GLOBAL_BASE_ADDR` to address 0 and passes every other address unchanged. The peripheral crossbar has a second initiator, the `SEP_LOCAL` port of the output fabric's route demux, so local masters reach the mailboxes and system CSRs through the same decode. Addresses outside the mailbox and system CSR windows but below `0x4000_0000` are truncated to 32 bits, their IDs reduced from 7 to 3 bits, and forwarded to the `ext` initiator of the local crossbar. Inside the system CSR windows the crossbar decodes each register block separately and answers the addresses between them with DECERR on reads and writes, as it does any address at or above `0x4000_0000`. The system CSRs program the inbound filter entries and the SEP aperture, as well as the output fabric's remaps and outbound filter.

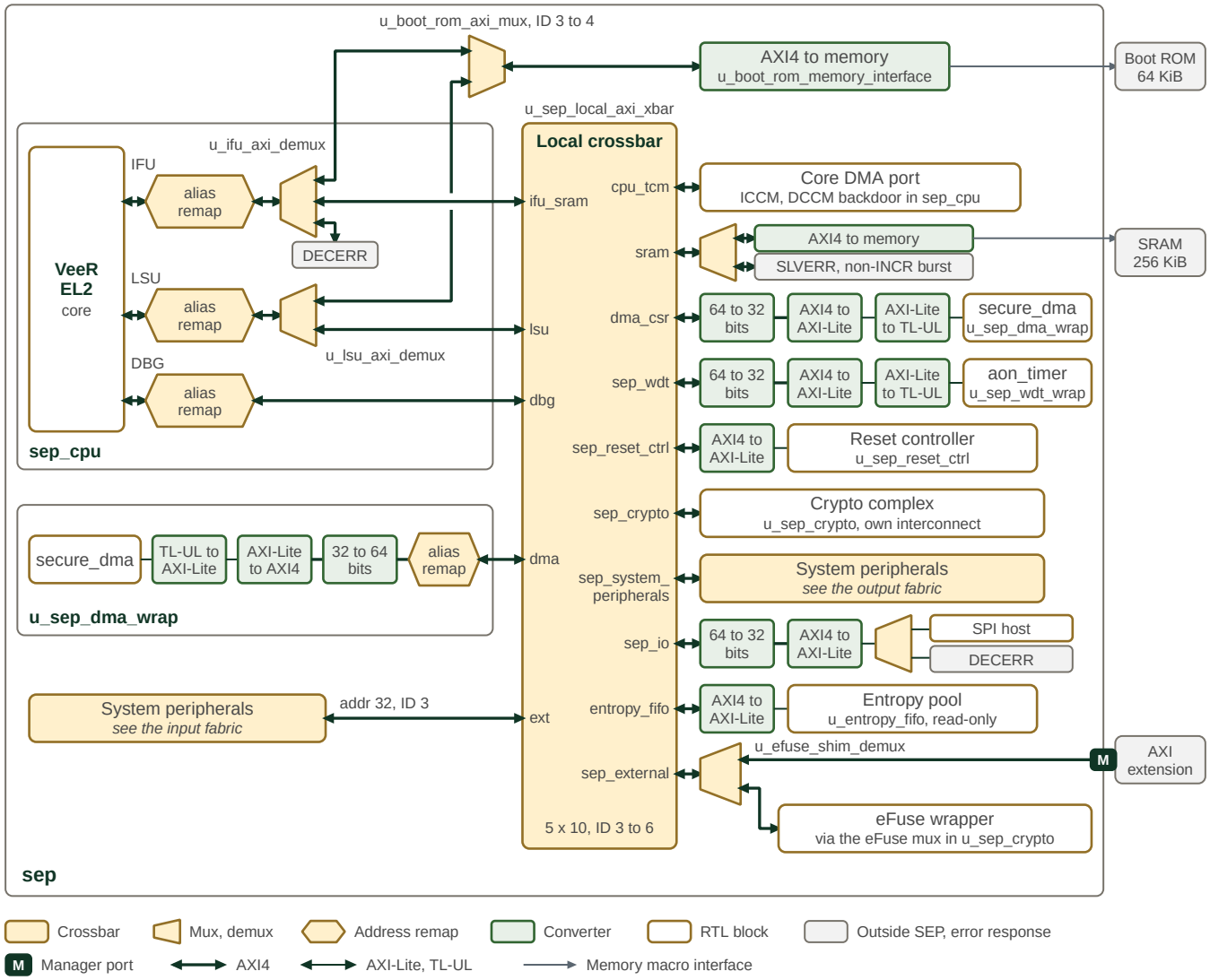


*Figure 17. SEP local fabric*

Figure 17 shows the local crossbar with its initiators on the left and its targets on the right, in crossbar port order. Each CPU master and the secure DMA master pass a local alias remap before the crossbar; the remap rewrites the 768 MiB window at `SEP_LOCAL_BASE_ADDR` to `0x1000_0000`. The IFU and LSU demuxes inside `sep_cpu` send boot ROM addresses to the boot ROM mux, which reaches the ROM macro without passing the crossbar, and the IFU demux answers addresses outside the boot ROM and SEP SRAM with DECERR. The chain on each target row is the conversion the target needs: the secure DMA registers and the watchdog are reached as 32-bit TL-UL, and the IO block, reset controller and entropy pool as AXI-Lite. The `sep_external` demux diverts the eFuse shim CSR window to the eFuse wrapper; every other address in the `0x2000_0000` to `0x3FFF_FFFF` aperture leaves SEP on the AXI extension port. Figure 15 gives the initiator-to-target connections.

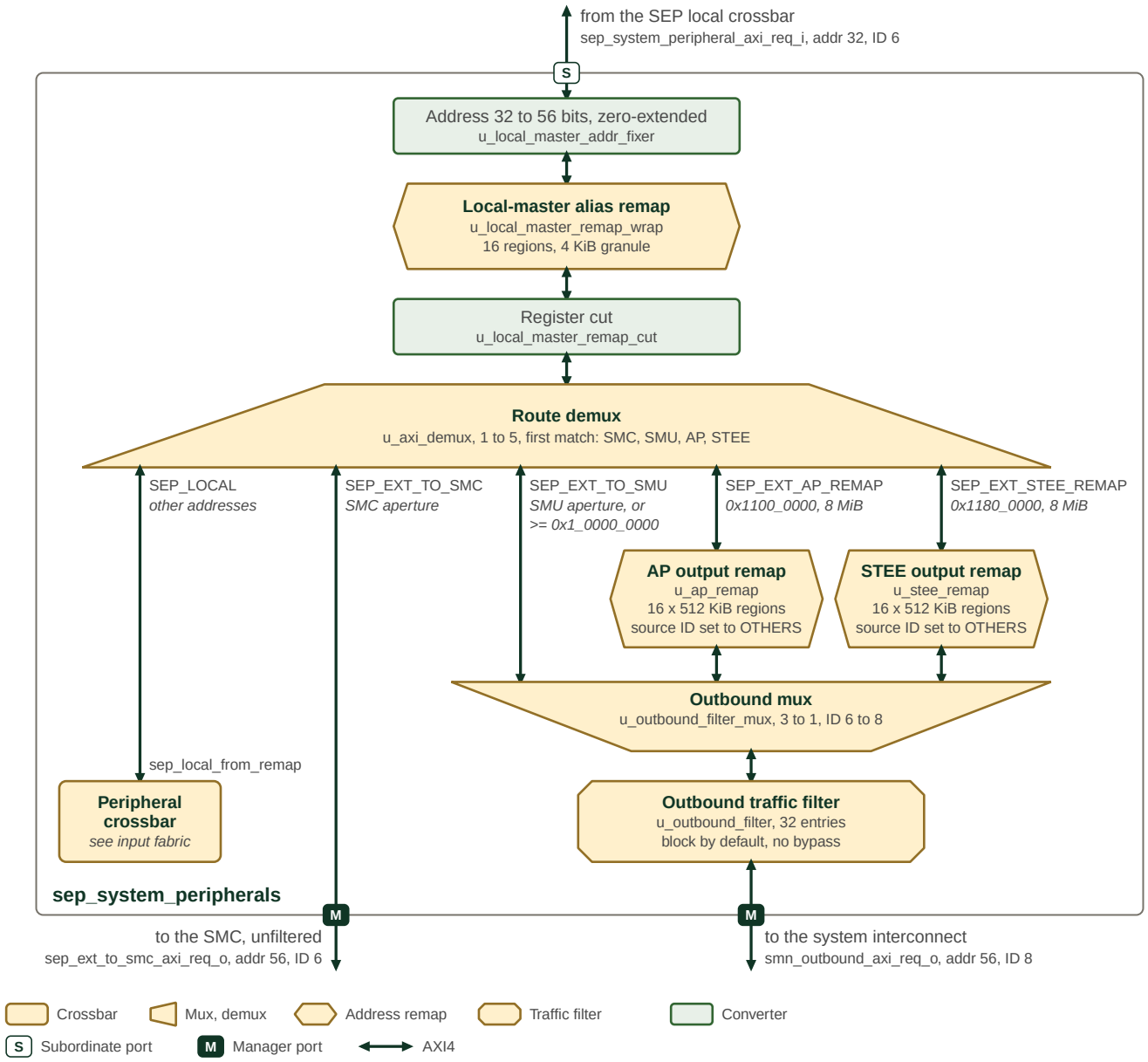


*Figure 18. SEP output fabric*

Figure 18 follows local-master traffic that the local crossbar sends to `sep_system_peripherals`. The address is zero-extended from 32 to 56 bits, passes the 16-region alias remap and a register cut, and reaches the route demux. The demux takes the first match in the order shown: the SMC aperture, supplied by the SMC, goes straight to the SMC without filtering; the SMU aperture from `SMU_GLOBAL_BASE_ADDR` and `SMU_REGION_SIZE`, and any address at or above `0x1_0000_0000`, goes to the outbound mux; the AP and STEE regions pass their output remaps, which replace the address with the programmed region offset and set the source ID to `OTHERS`, before the same mux; and every other address returns to the peripheral crossbar of the input fabric. The outbound mux widens the ID from 6 to 8 bits, and the outbound filter, which has no bypass, guards the path to the system interconnect.

### Interfaces

- Memory interfaces: BL0 ROM, DCCM, ICCM, Scratch SRAM, KM SRAM, KM ROM, OTBN IMEM, OTBN DMEM
- Fuse controller interface
- Entropy source interface

- JTAG interfaces
- *RAS Interface - Alarm Processor is deferred to next generation*

**Address Remapping**

Sixteen remap regions are available, each supported by an address remapper. Remappers define the following attributes:

- Input address range
- Remapped output address offset
- AxCACHE value, which replaces the transaction's own on a hit

The programmable remap regions accomplish several jobs. First, they expand local 32-bit addresses to the global 56-bit address space. Second, they provide software-programmed translation and cacheability attributes for locally initiated transactions. Out of reset, these programmable remap entries are invalid and therefore behave as transparent pass-through entries until firmware programs them.

After the alias remap, traffic routing is done as follows:

- External or cross-subsystem traffic reaches SEP through the global aperture described by `SEP_GLOBAL_BASE_ADDR` and `SEP_REGION_SIZE`. The current `SEP_REGION_SIZE` reset is `0x0100_0000` (16 MiB); the runtime value is software-programmed to match the chiplet map.
- SEP CPU local-alias traffic is an internal CPU-side view of the SEP map. It exists so software can use high alias addresses such as `0xD000_0000` for resources whose direct internal addresses begin at `0x1000_0000`. This high local alias span has a fixed SEP-local size of 768 MiB.
- Any transaction that doesn't get routed to local resources is directly routed out to the SMN.

Both the chiplet ID and chiplet BAR are available in SMC as MMRs. The Programmer's Guide gives the aperture programming sequence.

**Filtering and Protection**

Both SEP filters are instances of the common `axi_filter_wrap` block. The inbound filter has 16 entries and the outbound filter has 32; each entry is one `filter_ctrl` register triple (`FILTER_CONFIG`, `START_ADDR`, `END_ADDR`). Both carry 56-bit addresses on a 64-bit data bus, both enable NS and source ID matching, and both block traffic that matches no entry.

The inbound filter is bypassed in its entirety, admitting all inbound traffic, while SEP-scope debug is enabled; see Disable Vector Format (SIP_DIS / SYS_DIS) for the `SEP_DBG` control. The outbound filter has no bypass and is always in effect.

Register regions accept single-beat accesses only (Single-Beat Register Access), so an inbound entry covering one with `allow_burst` set admits bursts the target rejects. The Programmer's Guide gives the filter programming rules.

The decode algorithm — entry priority, the match conditions, the address range granule and its widening direction, and the response returned to a blocked initiator — is common to SEP and SMC. It is documented once with the IP in AXI Traffic Filter.

**Routing**

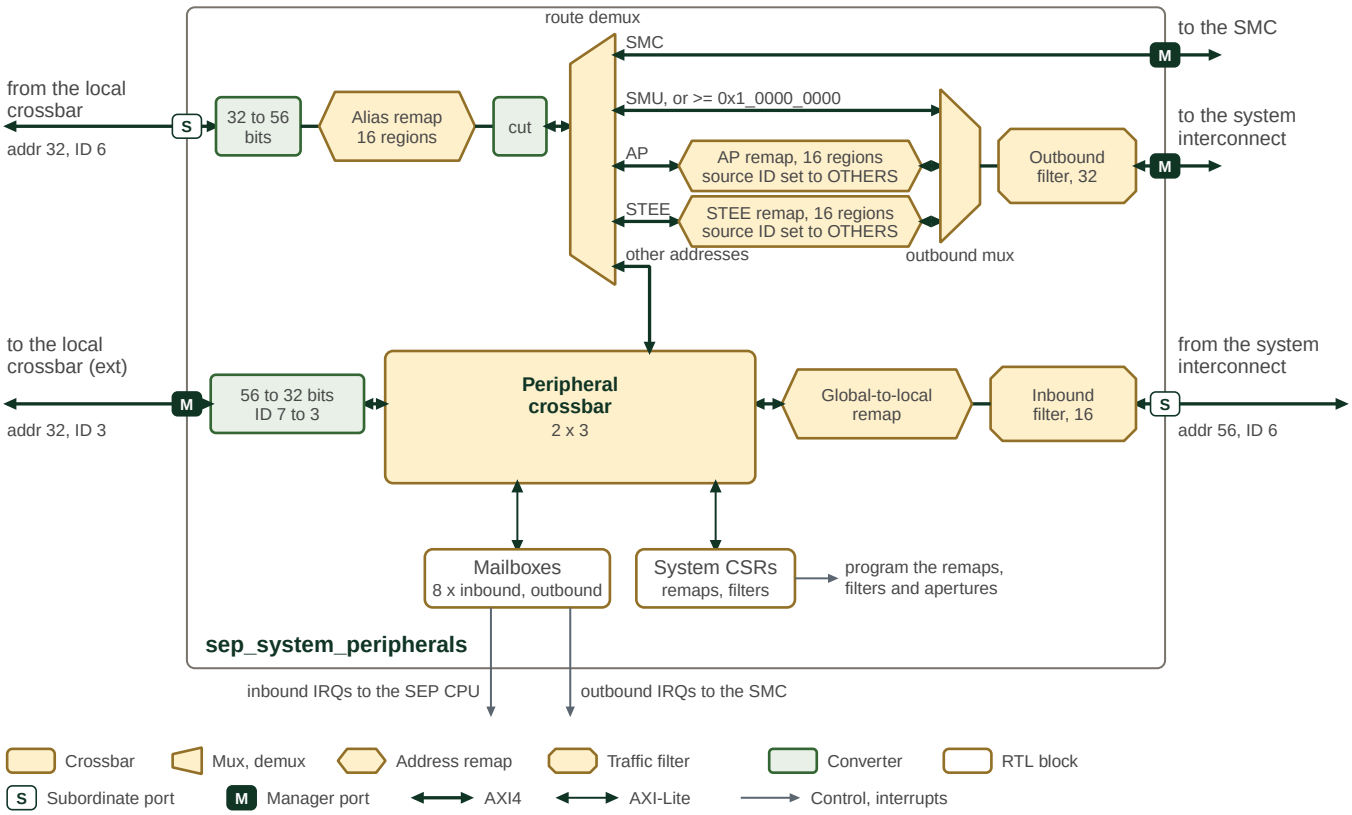


*Figure 19. SEP system-peripheral traffic management*

Figure 19 illustrates the AXI4 traffic management performed by the system peripherals block. Its left edge faces the SEP local crossbar and its right edge the SMU; the upper lane is the outbound path and the lower lane the inbound path, drawn in the notation of Fabric Topology. The figure is an overview: Figure 18 and Figure 16 give every element with its instance name and address window. The AXI4 subordinate port on the left is driven by the local masters that the local crossbar connects to it: the CPU LSU, the CPU debug master and the secure DMA. Traffic may be local to the SEP or outbound to external resources either within the SMU or beyond the SMU. Traffic beyond the SMU to the rest of the chiplet or remote chiplets will bridge to the system NOC also known as the SMN.

### Transaction Routing

Local SEP resources include peripheral CSRs, SRAM and the AP- and STEE-controlled remappers. SEP CPU direct addresses, the CPU local-alias view and the external global aperture are distinct views, as described in Address Remapping.

Neighboring SMC resources are accessed through the SMC global aperture programmed by SMC firmware and supplied to SEP/SMU integration logic. That aperture is independent from the SEP CPU local-alias range.

Traffic not routed to local SEP or neighboring SMC resources goes to the SMN system NoC, either through the AP- or STEE-controlled remappers, which external software such as the Ascalon AP controls, or through the direct SMN path.

SEP outbound traffic carries a source ID in AxUSER. Both AP- and STEE-controlled remappers replace it with `OTHERS`. Other traffic retains the initiator's ID: the CPU masters drive SEP's ID, while the secure DMA master leaves AxUSER at zero (`OTHERS`). AxPROT, including NS, passes unchanged; the initiators' values are in Table 14. The outbound filter checks address, NS and source ID after remapping. The Programmer's Guide describes how to select the outbound path and restrict `OTHERS` traffic.

The inbound filter blocks traffic by default. Its entries reside in the system CSRs, reached directly by local masters and by external masters only through an enabled inbound filter entry (or the stated SEP debug bypass). Matching rules are in Filtering and Protection.

See Mailboxes under Peripherals for host mailbox operation.

#### Memory Map and Register Reference

#### Memory Map

The Memory Map chapter lists the address windows. The common AXI Traffic Filter reference describes the filter register layout and programming rules.

Memory Map; AXI Traffic Filter.

#### Detailed Register Map

See the owning register references linked above.

### 2.1.5.8. Interrupts

#### Controllers

The SEP integrates a VeeR EL2 core with a built-in **Programmable Interrupt Controller (PIC)**. All maskable external interrupt sources feed a single `extintsrc_req` bus; the memory-mapped `meipl`, `meie`, and `meigwctrl` registers configure priority, enable, and gateway polarity per source.

The PIC base address is `0xC008_0000`; its register layout is described in the Register Interfaces section below.

#### Sources and Routing

The vector driven to the core, `sep_internal_interrupts`, is indexed from 0. Bit `i` of that vector is PIC source `i + 1`. The core's `extintsrc_req` port is numbered from 1, and the core ties source 0 to zero, so the low bit of the driven vector is the first programmable source. The comment directly above `NUM_INTERNAL_IRQS` in `sep_pkg.sv` numbers sources by that vector index, one less than the PIC source in Table 56.

#### SEP CPU Interrupt Vector Map

The PIC supports 255 external sources. Slots 1–43 are individually assigned to specific peripherals and subsystems connected to the SEP, including both SEP-internal blocks (crypto engines, DMA, entropy) and cross-block sources (SMC mailboxes, SPI). Slots 44–255 are reserved for chip-level external inputs passed in at the SEP block boundary.

Table 56 lists the assigned PIC sources and the range reserved for chip-level inputs.

*Table 56. SEP PIC source assignments*

| PIC source | Source | Description |
|---|---|---|
| 1 | Mailbox interrupt 0 | SMC-to-SEP mailbox channel 0 interrupt |
| 2 | Mailbox interrupt 1 | SMC-to-SEP mailbox channel 1 interrupt |
| 3 | Mailbox interrupt 2 | SMC-to-SEP mailbox channel 2 interrupt |
| 4 | Mailbox interrupt 3 | SMC-to-SEP mailbox channel 3 interrupt |
| 5 | Mailbox interrupt 4 | SMC-to-SEP mailbox channel 4 interrupt |

| PIC source | Source | Description |
|---|---|---|
| 6 | Mailbox interrupt 5 | SMC-to-SEP mailbox channel 5 interrupt |
| 7 | Mailbox interrupt 6 | SMC-to-SEP mailbox channel 6 interrupt |
| 8 | Mailbox interrupt 7 | SMC-to-SEP mailbox channel 7 interrupt |
| 9 | DMA transfer done | Secure DMA full-transfer-complete interrupt |
| 10 | DMA chunk done | Secure DMA chunk-transfer-complete interrupt |
| 11 | DMA error | Secure DMA error interrupt |
| 12 | DMA alert | Secure DMA aggregated alert (fault) |
| 13 | WDT alert | Watchdog timer aggregated alert. Note: the WDT **bark** goes to the NMI pin, not here; see Non-PIC Interrupt Paths. |
| 14 | SPI IRQ | SPI host controller interrupt |
| 15 | KM mailbox IRQ | Key Manager-to-SEP mailbox interrupt |
| 16 | Entropy source IRQ | OpenTitan entropy source interrupt |
| 17 | External TRNG IRQ | External TRNG interrupt; driven to zero in the current OSS integration |
| 18 | HMAC done | HMAC operation complete interrupt |
| 19 | HMAC FIFO empty | HMAC message FIFO empty interrupt |
| 20 | HMAC error | HMAC error interrupt |
| 21 | KMAC done | KMAC operation complete interrupt |
| 22 | KMAC FIFO empty | KMAC message FIFO empty interrupt |
| 23 | KMAC error | KMAC error interrupt |
| 24 | CSRNG command request done | CSRNG command request complete interrupt |
| 25 | CSRNG entropy request | CSRNG entropy request interrupt |
| 26 | CSRNG HW instance exception | CSRNG hardware instance exception interrupt |
| 27 | CSRNG fatal error | CSRNG fatal error interrupt |
| 28 | EDN command request done | Entropy Distribution Network command request complete interrupt |
| 29 | EDN fatal error | Entropy Distribution Network fatal error interrupt |
| 30 | OTBN done | OpenTitan Big Number Accelerator operation complete interrupt |
| 31 | KM unrecoverable error | Key Manager unrecoverable error interrupt; also exported as a SEP output port |
| 32 | KM recoverable error | Key Manager recoverable error interrupt; also exported as a SEP output port |
| 33 | Crypto alert | Combined OpenTitan crypto alert. SEP has no `alert_handler`; each accelerator has its own `prim_alert_receiver` channel(s) with ping disabled, and those pulses plus integrity-fails share this source. See OpenTitan Alert Termination. |
| 34 | Locked field access | eFuse locked-field access violation interrupt |
| 35 | Adams Bridge error | Adams Bridge (post-quantum accelerator) error interrupt |
| 36 | Adams Bridge notification | Adams Bridge (post-quantum accelerator) operation notification interrupt |
| 37 | Entropy pool low | Entropy pool FIFO low-water condition (informational: pool draining faster than filling) |
| 38 | Entropy pool fill stall | Entropy pool FIFO fill-path stall (fault: EDN not making progress) |
| 39 | Entropy pool pointer fault | Entropy pool FIFO read/write pointer integrity fault (fault). The pool's pointer-integrity check is not built in the current configuration, so this source is tied to zero and never asserts; the slot is assigned anyway so that enabling the check later does not renumber the map. When live it is level-sensitive and holds until a `TRNG_ENTROPY_CLEAR` pool scrub resets the pointers; there is no write-one-to-clear. |
| 40 | Token match fault | eFuse token comparator redundancy fault (fault). The OR of the three sticky `TOKEN_MATCH_FAULT` bits. Those bits are software-read-only and clear only on reset, so this source stays asserted; there is no write-one-to-clear. |

| PIC source | Source | Description |
|---|---|---|
| 41 | DMA register-path bus error | A CPU access to a Secure DMA register returned a TL-UL error response (unmapped or misaligned offset). The faulting access also received SLVERR synchronously. Latched and held until firmware writes `DMA_BUS_ERR_CLEAR.clr`. |
| 42 | DMA host-path integrity/bus fault | Either the fabric returned a non-OKAY AXI response to a DMA transfer, or a TL-UL command-integrity check failed on a command issued by the DMA. The integrity case is a fault-detection event with no other reporting path. Cleared by the same `DMA_BUS_ERR_CLEAR.clr` write. |
| 43 | Peripheral register-bridge fault | The OR of the per-block register-bridge error bits for AES, HMAC, KMAC, OTBN, CSRNG, EDN and the watchdog timer. `PERIPH_BUS_ERR_STATUS` identifies the block, and `PERIPH_BUS_ERR_CLEAR` clears one block's bit at a time, so acknowledging one cannot discard another block's fault. See Crypto Register-Bridge Faults. |
| [44..255] | Reserved (external band) | Chip-level external interrupt inputs |

### Non-PIC Interrupt Paths

Three CPU interrupt pins bypass the EL2 PIC entirely. The core also reports its own memory errors through a path that is neither a pin nor a PIC source; see Memory ECC Errors.

### NMI — Watchdog Timer Bark

The VeeR EL2 `nmi_int` pin is wired to the watchdog timer bark signal from `sep_wdt_wrap`. When the bark fires, the CPU jumps to the NMI vector address held in the `SEP_NMI_VEC` register. The `SEP_NMI_VEC_LOCK` register prevents runtime modification of the vector once locked.

The NMI is non-maskable: it cannot be suppressed by `mstatus.MIE` or any PIC configuration register.

### Timer and Software Interrupts

The `timer_int` and `soft_int` pins of the VeeR EL2 core are tied to zero at the SMU integration boundary. There is no external CLINT-style delivery path. The external pins drive `mip.MTIP` and `mip.MSIP`, respectively, so these pending bits remain inactive in this integration.

The two internal timers instead report `mip.MITIP0` (bit 29) and `mip.MITIP1` (bit 28). Enable them through `mie.MITIE0` and `mie.MITIE1` at the corresponding bit positions, together with `mstatus.MIE`. Their interrupt cause codes are `0x1D` and `0x1C`, respectively (with the interrupt bit set in `mcause`). Enabling `mie.MTIE` or `mie.MSIE` does not enable the internal timers.

### Memory ECC Errors

The ICCM and DCCM are SECDED-protected. Neither reaches the PIC: the core reports these errors itself, correctable ones as a local interrupt and uncorrectable ones as a synchronous trap. No PIC register affects either, and there is no PIC source to enable or mask.

The SEP core is configured without an I-cache, so the I-cache parity counter `MICECT` (`0x7F0`) never increments. Only the two tightly-coupled-memory counters below are live.

#### Correctable (single-bit) errors

Hardware corrects the data and the access is retried, so no data is lost and no trap is taken. Each protected memory has a counter CSR (Table 57).

*Table 57. SEP correctable-error counters*

| CSR | Address | Counts |
|---|---|---|
| `MICCMECT` | `0x7F1` | ICCM single-bit corrections |
| `MDCCMECT` | `0x7F2` | DCCM single-bit corrections |

Bits `[26:0]` are the running count. Bits `[31:27]` hold a threshold exponent: the counter raises a request once the count reaches $2^{threshold}$. A written threshold above 26 is clamped to 26. The requests from both counters are ORed into `mip[30]` (`MCEIP`) and delivered when `mie[30]` (`MCEIE`) and `mstatus.MIE` are both set, with `mcause` = `0x8000_001E`.

Counters and thresholds reset to zero, so a zero threshold raises the request on the first corrected error. `mie[30]` is clear at reset, and until it is set corrected errors are counted silently. `MCEIP` follows the threshold comparator and has no write-one-to-clear: it stays asserted until the count drops below the threshold. The memory error handling section of the Programmer's Guide gives the enable and service sequence.

#### Uncorrectable (double-bit) errors

These cannot be masked and have no enable bit. The access returned no valid data, so the core traps on the instruction that hit the error (Table 58).

*Table 58. SEP uncorrectable-error traps*

| Access | `mcause` | Trap |
|---|---|---|
| Instruction fetch from ICCM | `0x0000_0001` | Instruction access fault |
| Load from DCCM | `0x0000_0005` | Load access fault |
| Store to DCCM | `0x0000_0007` | Store access fault |
| DCCM access on the fast-interrupt path | `0xF000_1000` | NMI, taken through `SEP_NMI_VEC` like the watchdog bark |

`mcause` alone does not identify these as memory failures — a bad pointer and a failing SRAM both raise a load access fault. `mscause` (`0x7FF`) is what separates them: on an access fault, `mscause` = `1` means uncorrectable ECC. A handler that does not read `mscause` cannot distinguish a software bug from failing hardware.

The faulting access cannot be completed, and retrying it re-reads the same corrupted location.

An uncorrectable error on an externally initiated (DMA) access to either memory is not a CPU trap at all. It is returned as an error response on that transaction, and the initiator is responsible for reporting it.

#### Debug bus mirror

The ICCM and DCCM single- and double-bit error signals are also mirrored onto `ext_debug_bus_o[367:364]`, most significant first: ICCM single, ICCM double, DCCM single, DCCM double. These are single-cycle pulses with no latch and no handshake, provided for external trace and bring-up observability. They are not a delivery path, so software must not rely on sampling them; use the CSRs above.

#### Software Handling

Interrupt source IDs are **1-based** inside the PIC. Source 0 is permanently tied to zero inside the core. Firmware programs source `N` through the per-source arrays, each indexed at an offset of `N * 4` from its region base: `meipl` at PIC base `+0x0000`, `meie` at `+0x2000`, `meigwctrl` at `+0x4000` and `meigwclr` at `+0x5000`. Pending status is the exception — `meip` at `+0x1000` is a bitmap rather than a per-source array, so source `N` is bit `N % 32` of `meip[N / 32]`.

Each per-source array carries an entry 0, so that the array index is the source ID and no off-by-one applies: `meipl[7]` is source 7. That entry is reserved and has no hardware behind it — the PIC decodes the address but ties the register off, so `+0x0000`, `+0x2000`, `+0x4000` and `+0x5000` read as zero and discard writes. It appears in the RDL, the generated headers and

the UVM RAL only to preserve the indexing. The PIC section of the Programmer's Guide gives the programming and per-source service rules.

| | |
|---|---|
| **NOTE** | Register tests that walk the whole PIC map must skip index 0 of `meipl`, `meie`, `meigwctrl` and `meigwclr`. The register model declares those entries read-write like every other element, but the hardware treats them as reserved, so a read-write check on them fails. |

### Memory Map and Register Reference

#### Memory Map

The EL2 PIC register block (`meipl`, `meie`, `meigwctrl`, `meigwclr`, and related registers) is documented in the register reference below.

The per-source arrays in that table are listed from index 0. Index 0 is the reserved entry described in Interrupts: it exists so the array index equals the interrupt source ID, and the first programmable source is index 1.

#### Detailed Register Map

#### EL2_PIC

VeeR EL2 Programmable Interrupt Controller (PIC) memory-mapped registers. Supports up to PIC_TOTAL_INT external interrupt sources with 15 priority levels.

**NOTE**

**Register Arrays:**

- **meipl[256]**: 256 registers, base 0x0, stride 0x4
- **meip[8]**: 8 registers, base 0x1000, stride 0x4
- **meie[256]**: 256 registers, base 0x2000, stride 0x4
- **meigwctrl[256]**: 256 registers, base 0x4000, stride 0x4
- **meigwclr[256]**: 256 registers, base 0x5000, stride 0x4

*Table 59. EL2_PIC register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x3FC | meipl[256] | RW | External Interrupt Priority Level Register. Sets the priority level for an interrupt source. |
| 0x1000 - 0x101C | meip[8] | R | External Interrupt Pending Register. Reports pending status of up to 32 interrupt sources. |
| 0x2000 - 0x23FC | meie[256] | RW | External Interrupt Enable Register. Controls interrupt enable for a single source. |
| 0x3000 | mpiccfg | RW | PIC Configuration Register. Used to select the operational parameters of the PIC. |
| 0x4000 - 0x43FC | meigwctrl[256] | RW | External Interrupt Gateway Configuration Register. Controls interrupt type and polarity. |
| 0x5000 - 0x53FC | meigwclr[256] | W | External Interrupt Gateway Clear Register. Writing any value clears the gateway's interrupt pending (IP) bit for edge-triggered interrupts. |

### meipl[256]

*Table 60. el2_pic.meipl[256] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3:0 | `intpriority` | RW | 0x0 | External interrupt priority level: RISC-V standard order: 0=Never interrupt, 1..15=Priority (1 lowest, 15 highest) Reverse order: 15=Never interrupt, 14..0=Priority (14 lowest, 0 highest) |

### meip[8]

*Table 61. el2_pic.meip[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `intpend` | R | 0x0 | External interrupt pending bits. Bit Y indicates pending status for source X*32+Y. 0: Interrupt not pending 1: Interrupt pending |

### meie[256]

*Table 62. el2_pic.meie[256] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `inten` | RW | 0x0 | External interrupt enable: 0: Interrupt disabled 1: Interrupt enabled |

### mpiccfg

*Table 63. el2_pic.mpiccfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `priord` | RW | 0x0 | Priority order: 0: RISC-V standard compliant priority order (0=lowest to 15=highest) 1: Reverse priority order (15=lowest to 0=highest) |

### meigwctrl[256]

*Table 64. el2_pic.meigwctrl[256] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `irq_type` | RW | 0x0 | External interrupt type: 0: Level-triggered interrupt 1: Edge-triggered interrupt |
| 0 | `polarity` | RW | 0x0 | External interrupt polarity: 0: Active-high interrupt 1: Active-low interrupt |

### meigwclr[256]

*Table 65. el2_pic.meigwclr[256] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `clear` | W | 0x0 | Write any value to clear the interrupt pending bit in the gateway. This register is write-only; reads return 0. |

### 2.1.5.9. Peripherals

SEP peripherals provide data movement, mailboxes, external I/O and watchdog services. Reset control is described in Clocks and Resets; OTP and lifecycle controls are described in Security and Lifecycle.

#### Direct Memory Access (DMA)

The SEP DMA copies data between addressable memories and peripheral registers, with optional inline SHA-256, SHA-384 or SHA-512 hashing, and fills memory with zeros. It uses the OpenTitan DMA Controller inside an AXI wrapper, with the zero fill added. The integration below defines which upstream ports and hardware triggers are available in SEP. The Programmer's Guide gives the programming guidance for this block.

Source and destination addresses independently support incrementing, fixed address and wrap-at-chunk-boundary operation. Use incrementing addresses for memory and a fixed address for a peripheral FIFO. Transfers use 1-, 2- or 4-byte accesses; inline hashing requires 4-byte transfers. The core issues individual TL-UL transactions, so the AXI wrapper does not turn a copy into a burst transfer. Hash results are read from the DMA digest registers.

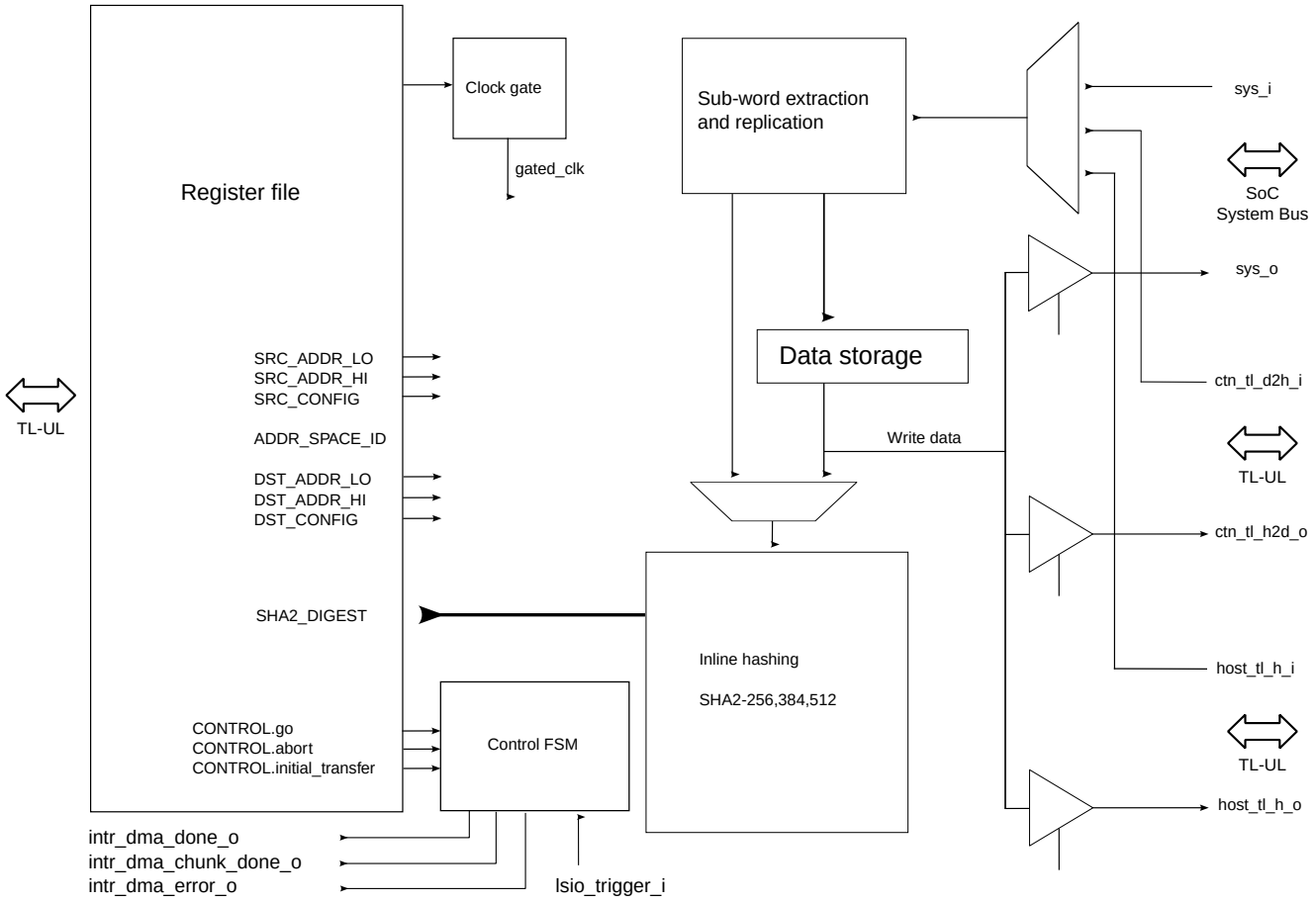


*Figure 20. OpenTitan DMA core and inline hashing reference architecture*

Figure 20 shows the upstream DMA core. SEP adapts the host register interface to its AXI fabric as described below.

#### OCAH Modifications

Figure 21 shows how `sep_dma_wrap` connects the DMA core to the SEP local crossbar. The register path runs left to right along the top and the host path right to left along the bottom; green boxes are converters, and the dashed ports are tied

off.

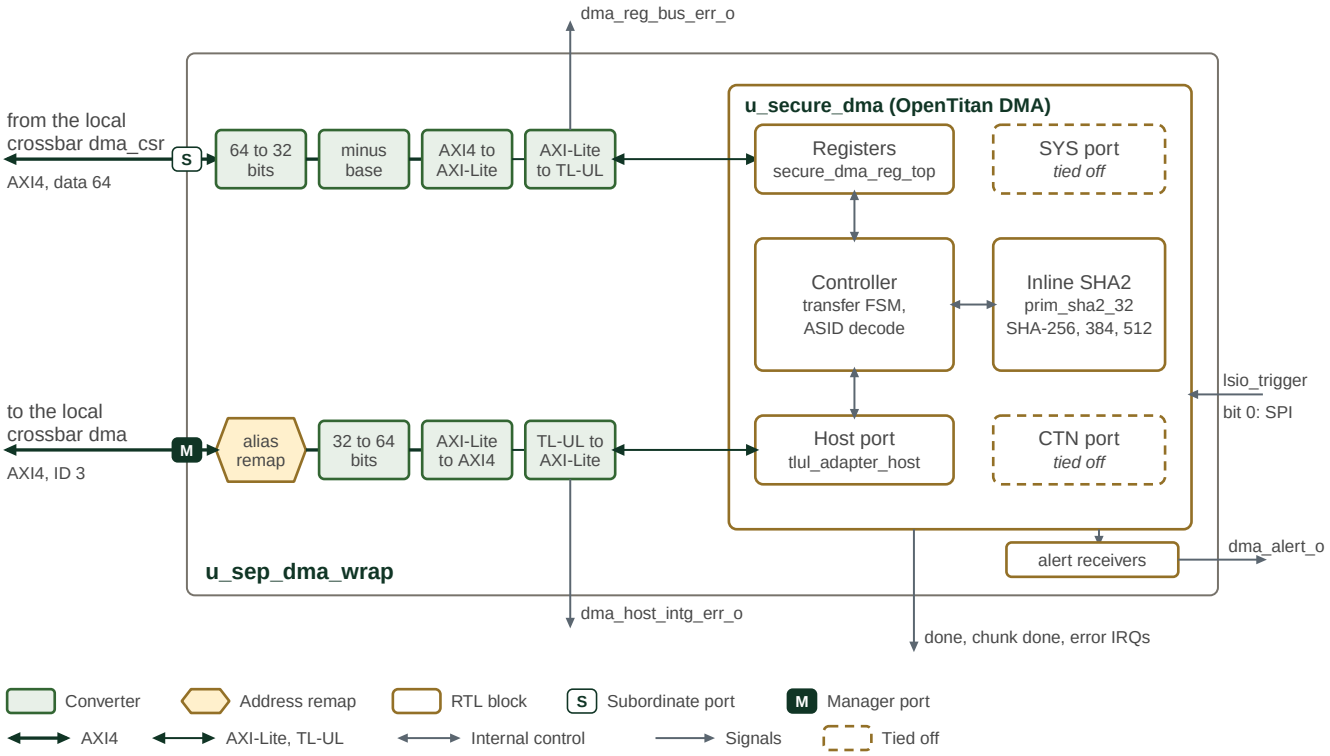


*Figure 21. SEP secure DMA integration*

The wrapper exposes separate AXI4 register-slave and DMA-master interfaces, both with 32-bit addresses and 64-bit data. Internally, the register path converts AXI4 to 32-bit AXI4-Lite and then TL-UL; the data path converts the core's TL-UL host port back to AXI4. A local-alias remapper precedes the SEP crossbar and leaves the CPU TCM backdoor range available to DMA.

Address-space ID `0x7` selects the core's internal host port, the only route in this wrapper. The upstream SoC control and system ports are tied off; their ASIDs do not provide additional SEP routes. Reachability is governed by the SEP fabric and filters.

The master path leaves AxUSER at zero and drives AxPROT `0b001` (privileged, secure, data), the value the SEP CPU's LSU drives; TL-UL carries no protection attributes, so the wrapper sets it. The register and host bridges latch faults on `dma_reg_bus_err_o` and `dma_host_intg_err_o`, cleared through `DMA_BUS_ERR_CLEAR`; local alert receivers aggregate the core alerts into `dma_alert_o`. Selecting the tied-off CTN or SYS address space does not complete.

Only handshake input 0 is connected, to the SPI host's LSIO trigger. All other trigger inputs are tied low; SEP does not wire UART or I3C handshake paths. This does not prevent software-initiated accesses to reachable peripheral registers.

### Configuration Taken at Each Start

A start is a write of `CONTROL.GO` while `STATUS.BUSY` is clear: the first chunk of a transfer, with `INITIAL_TRANSFER` set, or a later chunk, with `GO` alone. The engine takes `OPCODE` and `DIGEST_SWAP` at the first start of a transfer and keeps them for its later chunks. It takes `HARDWARE_HANDSHAKE_ENABLE`, `RANGE_VALID`, `ENABLED_MEMORY_RANGE_BASE` and `ENABLED_MEMORY_RANGE_LIMIT` at every start, so a range narrowed or locked between chunks bounds the next chunk. The upstream DMA takes all of them at the first start only.

### Zero Fill

`CONTROL.OPCODE` 4 (`ZERO`) is an operation the upstream DMA does not have. A zero fill writes zeros to `TOTAL_DATA_SIZE` bytes from the destination address and never reads the source, so erasing memory needs no buffer of zeros that software prepares and that another initiator could change during the fill. Destination address modes, transfer widths, chunking, completion and error reporting, interrupts and `CONTROL.ABORT` behave as they do for a copy.

- The fill writes only inside the enabled memory range, from `ENABLED_MEMORY_RANGE_BASE` to `ENABLED_MEMORY_RANGE_LIMIT` inclusive, whatever the address spaces. The engine checks each write before it issues it, so the bound holds across chunks and in every address mode. The first write that would leave the range is not issued: the fill stops with `STATUS.ERROR` and `ERROR_CODE.DST_ADDR_ERROR`, and the writes before it remain. Every start takes the range again (Configuration Taken at Each Start), so locking it with `RANGE_REGWEN`, also between chunks, fixes this erase window until reset. The range does not bound a copy between internal addresses.
- The source registers are not used, but they are validated as for a copy, so program `SRC_ADDR_LO` and `SRC_ADDR_HI` as 0 with source address-space ID `0x7`.
- A zero fill cannot run in hardware handshake mode. With `HARDWARE_HANDSHAKE_ENABLE` set at any start, a later chunk included, it ends in `ERROR_CODE.OPCODE_ERROR` when the first trigger arrives.
- When the size is not a multiple of the transfer width, the last write covers only the remaining bytes.
- A zero fill writes no byte past `TOTAL_DATA_SIZE`. A start with no bytes left, such as `GO` alone after the last chunk, ends in `ERROR_CODE.SIZE_ERROR` without a write.
- ICCM and DCCM take only whole 32-bit words from the DMA. A zero fill there needs 4-byte transfers, a 4-byte-aligned destination and a size that is a multiple of 4; any other write ends in `ERROR_CODE.BUS_ERROR` and sets the host-path fault in `DMA_BUS_ERR_STATUS`.
- The internal register that carries a word from the read to the write of a copy holds zero after a zero fill, so the engine keeps no word of an earlier copy.

### Completion Reporting

The core reports transaction completion, chunk completion and error separately. Chunk completion applies to multi-chunk transfers without hardware handshake; it is not a per-trigger interrupt in handshake mode. Status completion and error bits are write-one-to-clear; the detailed register map defines each bit. Inline hashing requires 4-byte transfers, and `DIGEST_SWAP` controls byte order within each digest register, not register order. The SEP DMA section of the Programmer's Guide gives the copy, hashing, handshake and abort sequences.

### Future Enhancements

The following are proposals, not capabilities of the current SEP integration:

- Inline AES encryption or decryption (with/without inline hashing)
- Inline hashing with only digest written to target slave
- Optimize implementation by using cryptographic resources (AES, HMAC-SHA2) engines inside the cryptographic subsystem and adding streaming interfaces
- Replace the FSM controller with a microsequencer or simple CPU, potentially shared with the Key Manager

### Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

### DMA Controller

DMA Controller for the integrated OpenTitan.

**NOTE**

**Register Arrays:**

- **SHA2_DIGEST_0[16]**: 16 registers, base 0x58, stride 0x4
- **INTR_SRC_ADDR_0[11]**: 11 registers, base 0xA4, stride 0x4
- **INTR_SRC_WR_VAL_0[11]**: 11 registers, base 0x124, stride 0x4

*Table 66. DMA Controller register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | R | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | SRC_ADDR_LO | RW | - |
| 0x14 | SRC_ADDR_HI | RW | - |
| 0x18 | DST_ADDR_LO | RW | - |
| 0x1C | DST_ADDR_HI | RW | - |
| 0x20 | ADDR_SPACE_ID | RW | - |
| 0x24 | ENABLED_MEMORY_RANGE_BASE | RW | - |
| 0x28 | ENABLED_MEMORY_RANGE_LIMIT | RW | - |
| 0x2C | RANGE_VALID | RW | - |
| 0x30 | RANGE_REGWEN | RW | - |
| 0x34 | CFG_REGWEN | R | - |
| 0x38 | TOTAL_DATA_SIZE | RW | - |
| 0x3C | CHUNK_DATA_SIZE | RW | - |
| 0x40 | TRANSFER_WIDTH | RW | - |
| 0x44 | CONTROL | RW | - |
| 0x48 | SRC_CONFIG | RW | - |
| 0x4C | DST_CONFIG | RW | - |
| 0x50 | STATUS | RW | - |
| 0x54 | ERROR_CODE | R | - |
| 0x58 - 0x94 | SHA2_DIGEST_0[16] | R | - |
| 0x98 | HANDSHAKE_INTR_ENABLE | RW | - |
| 0x9C | CLEAR_INTR_SRC | RW | - |
| 0xA0 | CLEAR_INTR_BUS | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0xA4 - 0xCC | INTR_SRC_ADDR_0[11] | RW | - |
| 0x124 - 0x14C | INTR_SRC_WR_VAL_0[11] | RW | - |

### INTR_STATE

*Table 67. secure_dma.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `DMA_ERROR` | R | 0x0 | DMA error has occurred. DMA_STATUS.error_code register shows the details. |
| 1 | `DMA_CHUNK_DONE` | R | 0x0 | Indicates the transfer of a single chunk has been completed. |
| 0 | `DMA_DONE` | R | 0x0 | DMA operation has been completed. |

### INTR_ENABLE

*Table 68. secure_dma.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `DMA_ERROR` | RW | 0x0 | Enable interrupt when !!INTR_STATE.dma_error is set. |
| 1 | `DMA_CHUNK_DONE` | RW | 0x0 | Enable interrupt when !!INTR_STATE.dma_chunk_done is set. |
| 0 | `DMA_DONE` | RW | 0x0 | Enable interrupt when !!INTR_STATE.dma_done is set. |

### INTR_TEST

*Table 69. secure_dma.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `DMA_ERROR` | W | 0x0 | Write 1 to force !!INTR_STATE.dma_error to 1. |
| 1 | `DMA_CHUNK_DONE` | W | 0x0 | Write 1 to force !!INTR_STATE.dma_chunk_done to 1. |
| 0 | `DMA_DONE` | W | 0x0 | Write 1 to force !!INTR_STATE.dma_done to 1. |

### ALERT_TEST

*Table 70. secure_dma.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `FATAL_FAULT` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### SRC_ADDR_LO

*Table 71. secure_dma.SRC_ADDR_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `SRC_ADDR_LO` | RW | 0x0 | Lower 32 bits of the source address. Must be aligned to the transfer width. |

### SRC_ADDR_HI

*Table 72. secure_dma.SRC_ADDR_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `SRC_ADDR_HI` | RW | 0x0 | Upper 32 bits of the physical or virtual address of memory location within SoC memory address map or physical address within OT non-secure memory space. Must be aligned to the transfer width. Source and destination address must have the same alignment. |

### DST_ADDR_LO

*Table 73. secure_dma.DST_ADDR_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DST_ADDR_LO` | RW | 0x0 | Lower 32 bits of the destination address. Must be aligned to the transfer width. Source and destination address must have the same alignment. |

### DST_ADDR_HI

*Table 74. secure_dma.DST_ADDR_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DST_ADDR_HI` | RW | 0x0 | Upper 32 bits of the physical or virtual address of memory location within SoC memory address map or physical address within OT non-secure memory space. Must be aligned to the transfer width. Source and destination address must have the same alignment. |

### ADDR_SPACE_ID

*Table 75. secure_dma.ADDR_SPACE_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:4 | `DST_ASID` | RW | 0x7 | Target address space that the destination address pointer refers to. |
| 3:0 | `SRC_ASID` | RW | 0x7 | Target address space that the source address pointer refers to. |

### ENABLED_MEMORY_RANGE_BASE

*Table 76. secure_dma.ENABLED_MEMORY_RANGE_BASE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `BASE` | RW | 0x0 | Base Address to mark the start of the DMA enabled memory range within the OT internal memory space. |

### ENABLED_MEMORY_RANGE_LIMIT

*Table 77. secure_dma.ENABLED_MEMORY_RANGE_LIMIT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `LIMIT` | RW | 0x0 | Limit Address to mark the end of the DMA enabled memory range within the OT internal memory space; inclusive. |

### RANGE_VALID

*Table 78. secure_dma.RANGE_VALID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `RANGE_VALID` | RW | 0x0 | Once set the enabled memory base and limit registers are valid. |

### RANGE_REGWEN

*Table 79. secure_dma.RANGE_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3:0 | `REGWEN` | RW0C | 0x6 | Used by firmware to lock the DMA enabled memory range configuration registers from further modification. Once this register is set to kMultiBitBool4False, it can only be set to kMultiBitBool4True through a reset event.<br><br>Default Value = kMultiBitBool4True → Unlocked at reset. |

### CFG_REGWEN

*Table 80. secure_dma.CFG_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3:0 | `REGWEN` | R | 0x6 | Used by hardware to lock the DMA configuration registers. This register is purely managed by hardware and only software readable. |

### TOTAL_DATA_SIZE

*Table 81. secure_dma.TOTAL_DATA_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA_SIZE` | RW | 0x0 | Total size (in bytes) of the data to be transferred. The complete transfer operation may consist of multiple chunks of data as specified by the CHUNK_DATA_SIZE register.<br><br>Minimum: 1 byte. Maximum: May be restricted to a maximum pre-defined size based on OT DMA enabled memory space allocation. Works in conjunction with the TRANSFER_WIDTH register. |

### CHUNK_DATA_SIZE

*Table 82. secure_dma.CHUNK_DATA_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA_SIZE` | RW | 0x0 | Size (in bytes) for a single DMA transfer. In hardware handshake mode, the DMA reads in chunks of CHUNK_DATA_SIZE from the peripheral. For a single memory transfer CHUNK_DATA_SIZE and TOTAL_DATA_SIZE are set to the same value.<br><br>Minimum: 1 byte. Maximum: May be restricted to a maximum pre-defined size based on OT DMA enabled memory space allocation. Works in conjunction with the TRANSFER_WIDTH register. |

### TRANSFER_WIDTH

*Table 83. secure_dma.TRANSFER_WIDTH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1:0 | `TRANSACTION_WIDTH` | RW | 0x2 | Denotes the width of each transaction that the DMA shall issue during the data movement.<br><br>Multiple transactions of this width will be issued until TOTAL_DATA_SIZE bytes have been transferred. Note that firmware may need to set a different value if a receiving IP supports a read / write transaction width that is less than 1 DWORD. This does not affect the wrap-around mechanism. Note that the value 3 for this register represents an invalid configuration that leads to an error. |

### CONTROL

*Table 84. secure_dma.CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `GO` | RW | 0x0 | Setting this bit triggers the DMA operation. For normal operation, the DMA engine clears the `go` bit automatically after the configured operation is complete. For Hardware handshake operation, DMA engine does not auto clear the Go bit. Firmware shall clear the Go bit when it intends to stop the hardware handshake operation. |
| 30:28 | `Reserved` | - | - | Reserved |
| 27 | `ABORT` | W | 0x0 | Aborts the DMA operation if this bit is set. Sets the corresponding bit in the status register once abort operation is complete. Any OpenTitan-internal transactions are guaranteed to complete, but there are no guarantees on the SoC interface. |
| 26:9 | `Reserved` | - | - | Reserved |
| 8 | `INITIAL_TRANSFER` | RW | 0x0 | Marks the initial transfer to initialize the DMA and SHA engine for one transfer that can span over multiple single DMA transfers. Used for hardware handshake and ordinary transfers, in which multiple transfers contribute to a final digest. Note, for non-handshake transfers with inline hashing mode enabled, this bit must be set to also mark the first transfer. |
| 7:6 | `Reserved` | - | - | Reserved |
| 5 | `DIGEST_SWAP` | RW | 0x0 | Digest register byte swap.<br><br>If 1 the value in each digest output register is converted to big-endian byte order. This setting does not affect the order of the digest output registers, !!SHA2_DIGEST_0 still contains the first 4 bytes of the digest. |
| 4 | `HARDWARE_HANDSHAKE_ENABLE` | RW | 0x0 | Enable hardware handshake mode. Used to clear FIFOs from low speed IO peripherals receiving data, e.g., I3C receive buffer. Listen to an input trigger signal. Read data from source address location. Copy to destination address. Number of bytes specified in size register. Note assumption is the peripheral lowers input once FIFO is cleared. No explicit clearing necessary. |
| 3:0 | `OPCODE` | RW | 0x0 | Defines the type of DMA operations. |

### SRC_CONFIG

*Table 85. secure_dma.SRC_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `WRAP` | RW | 0x0 | When 0: Chunks occupy contiguous ascending addresses. When 1: Source address wraps back to the starting address when finishing a chunk. |
| 0 | `INCREMENT` | RW | 0x0 | Defines the increment behavior after every DMA read. When 0: Source address is not changed. All reads are done from the same address. When 1: Source address is incremented by transfer_width after each read. |

### DST_CONFIG

*Table 86. secure_dma.DST_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `WRAP` | RW | 0x0 | When 0: Chunks occupy contiguous ascending addresses. When 1: Destination address wraps back to the starting address when finishing a chunk. |
| 0 | `INCREMENT` | RW | 0x0 | Defines the increment behavior after every DMA write. When 0: Destination address is not changed. All writes are done to the same address. When 1: Destination address is incremented by transfer_width after each write. |

### STATUS

*Table 87. secure_dma.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `CHUNK_DONE` | RW1C | 0x0 | Transfer of a single chunk is complete. Only raised for multi-chunk memory-to-memory transfers. Cleared automatically by the hardware when starting the transfer of a new chunk. |
| 4 | `SHA2_DIGEST_VALID` | R | 0x0 | Indicates whether the SHA2_DIGEST register contains a valid digest. This value is cleared on the initial transfer and set when the digest is written. |
| 3 | `ERROR` | RW1C | 0x0 | Error occurred during the operation. ERROR_CODE register denotes the source of the error. |
| 2 | `ABORTED` | RW1C | 0x0 | Set once aborted operation drains. |
| 1 | `DONE` | RW1C | 0x0 | Configured DMA operation is complete. Cleared automatically by the hardware when starting a new transfer. |
| 0 | `BUSY` | R | 0x0 | DMA operation is active if this bit is set. DMA engine clears this bit when operation is complete. This bit may be set as long as hardware handshake mode is active and triggered. |

### ERROR_CODE

*Table 88. secure_dma.ERROR_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `ASID_ERROR` | R | 0x0 | The source or destination ASID contains an invalid value. |
| 6 | `RANGE_VALID_ERROR` | R | 0x0 | The DMA enabled memory range is not configured. |
| 5 | `BASE_LIMIT_ERROR` | R | 0x0 | The base and limit addresses contain an invalid value. |
| 4 | `BUS_ERROR` | R | 0x0 | The bus transfer returned an error. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3 | SIZE_ERROR | R | 0x0 | TRANSFER_WIDTH encodes an invalid value, TOTAL_DATA_SIZE or CHUNK_SIZE are zero, or inline hashing is not using 32-bit transfer width |
| 2 | OPCODE_ERROR | R | 0x0 | Opcode is invalid. |
| 1 | DST_ADDR_ERROR | R | 0x0 | Destination address is invalid. |
| 0 | SRC_ADDR_ERROR | R | 0x0 | Source address is invalid. |

### SHA2_DIGEST_0[16]

*Table 89. secure_dma.SHA2_DIGEST_0[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA_0 | R | 0x0 | SHA2 digest data |

### HANDSHAKE_INTR_ENABLE

*Table 90. secure_dma.HANDSHAKE_INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | Reserved | - | - | Reserved |
| 10:0 | MASK | RW | 0x7FF | Enable bits for incoming handshake interrupt wires. |

### CLEAR_INTR_SRC

*Table 91. secure_dma.CLEAR_INTR_SRC fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | Reserved | - | - | Reserved |
| 10:0 | SOURCE | RW | 0x0 | Source N needs interrupt cleared |

### CLEAR_INTR_BUS

*Table 92. secure_dma.CLEAR_INTR_BUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | Reserved | - | - | Reserved |
| 10:0 | BUS | RW | 0x0 | Bus selection bit for source N. |

### INTR_SRC_ADDR_0[11]

*Table 93. secure_dma.INTR_SRC_ADDR_0[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR_0 | RW | 0x0 | Destination address for interrupt source clearing write. |

### INTR_SRC_WR_VAL_0[11]

*Table 94. secure_dma.INTR_SRC_WR_VAL_0[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | WR_VAL_0 | RW | 0x0 | Write value for interrupt clearing write. |

### Mailboxes

Figure 22 shows the host mailbox connections and interrupt routing.

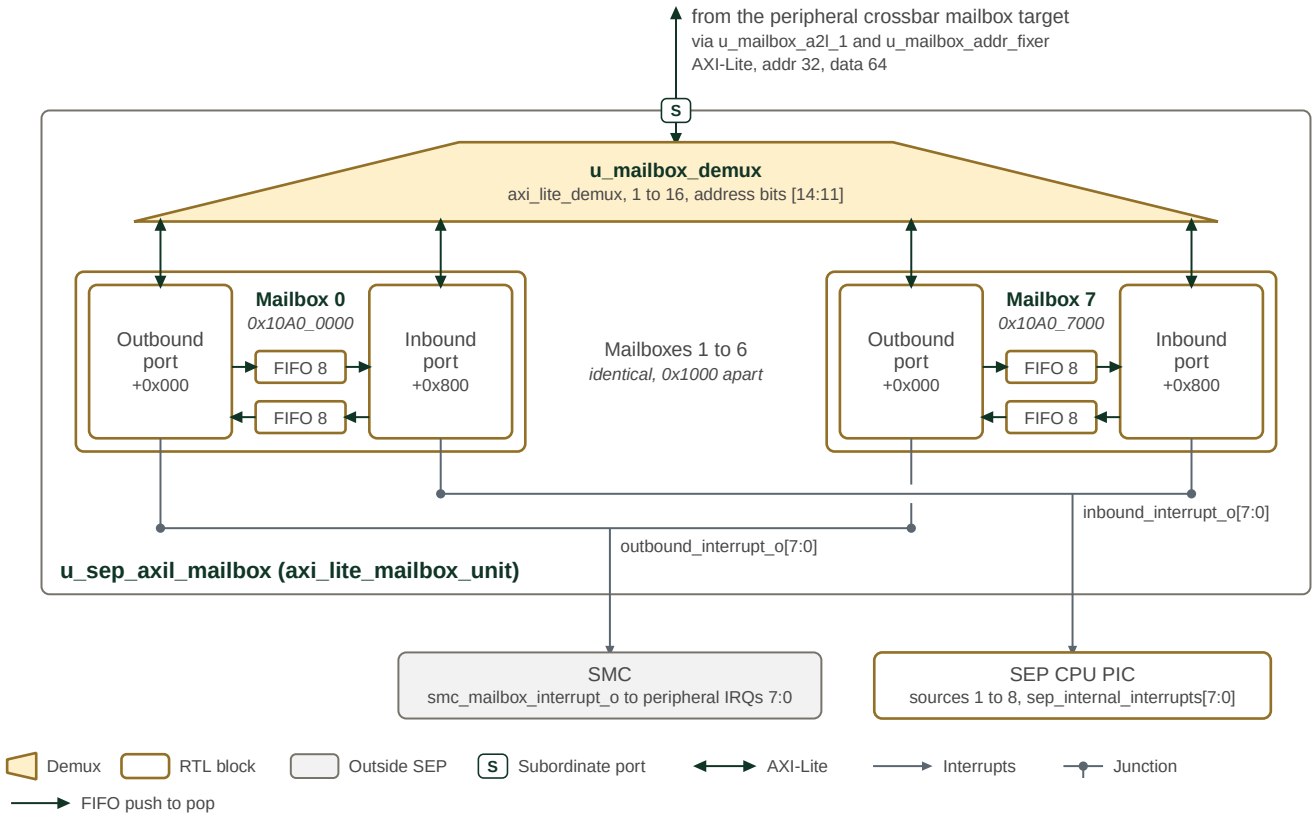


*Figure 22. SEP host mailbox architecture and interrupt routing*

The mailbox unit takes one AXI-Lite port from the peripheral crossbar, and its demux selects one of sixteen 2 kB port apertures from address bits 14 to 11. Mailbox $m$ occupies the 4 kB at `0x10A0_0000` + $m$ × `0x1000`, with its outbound port at offset `0x000` and its inbound port at offset `0x800`. The figure draws mailboxes 0 and 7; mailboxes 1 to 6 are identical. Each mailbox holds two FIFOs of eight 64-bit entries, and `WRITE_DATA` in one port pushes the FIFO that `READ_DATA` in the other port pops. Every port has its own level-sensitive, active-high interrupt. The eight inbound-port interrupts are SEP CPU PIC sources 1 to 8. The eight outbound-port interrupts leave SEP on `smc_mailbox_interrupt_o` and reach the SMC as its peripheral interrupts 7 to 0.

SEP uses mailboxes to exchange service requests with system clients, such as software running on application-processor (AP) cores. A client is the requester whose communication context software wants to keep separate; it need not be an entire processor. The system's firmware policy assigns mailbox apertures to clients and configures the inbound traffic filter to control their access. The hardware does not establish a fixed one-client-per-core allocation.

Each mailbox occupies a 4 KiB aperture and has an inbox/outbox pair. Its privacy depends on the programmed access policy, not on a distinct hardware mailbox type. When the inbox is written, an interrupt is sent to the SEP CPU to notify it of an incoming service request. Similarly, when the outbox is written by the SEP, an interrupt is sent to the SMC. The outbox is only used for communication with AP cores. For all other resources we only use inboxes on both ends of the transaction.

The SEP mailboxes have 64-bit data registers while SEP is a 32-bit system. Registers sit on an 8-byte stride and are decoded by which 8-byte slot the address falls in, so a register's low and high halves alias to the same register. The SEP CPU is RV32 and can reach only one half per instruction, which makes every half-word access a full-register access at the mailbox:

- A read of `READ_DATA` at either half pops one FIFO entry. The mailbox drives the whole entry; the CPU keeps the half selected by address bit 2 and discards the other. Reading both halves consumes two entries.
- A write of `WRITE_DATA` at either half pushes one entry, with bytes outside the write strobe pushed as zero. A 32-bit store to the low half enqueues `{32'h0, data}`.

SEP can use either half, but never both halves of one entry, so a SEP-visible payload is 32 bits per entry and both ends must agree on which half carries it. SEP firmware uses the low half at the register base offset.

**NOTE** This depends on the mailbox aperture staying side-effect in `mrac`. Clearing side-effect enables the core's load-to-load forwarding and speculative accesses, both of which pop entries unpredictably. The Programmer's Guide lists it among the mailbox usage rules.

Mailbox type is a configuration, not a hardware variant. Every mailbox is an identical instance of the same block; what makes one an ISS mailbox and another an STEE or Regular mailbox is the inbound traffic filter entry that covers its 4 kB aperture. The filter blocks by default, so a mailbox with no enabled entry over its aperture is unreachable from the fabric, and reprogramming an entry retypes that mailbox at run time. An entry pairs an address range with the qualification criteria below, plus separate read and write permissions and a write-once lock bit. See Filtering and Protection for the SEP filter instances and AXI Traffic Filter for the entry registers and the decode algorithm used to program them.

Table 95 describes the traffic qualifications used to distinguish mailbox types.

*Table 95. SEP mailbox qualification*

| Mailbox Type | Description |
|---|---|
| ISS | Uses fabric source ID to qualify incoming transactions. Legal source ID is programmable, lockable and reset by SEP reset. |
| STEE | Relies on STEE indicator in the transactions (NS bit) to qualify incoming transactions. |
| DTEE | Reserved. Would rely on the DTEE indicator in the transactions (confidential bit) to qualify incoming transactions. TODO: Confidential bit will be defined in future revisions, right now there is no hardware to qualify this. |
| Regular | Relies on PMA/PMP or page table for protection. No special action required by SEP. |

### Peripheral Subsystem

SEP reaches the serial-I/O peripherals in SMC over a dedicated AXI4 port, `sep_ext_to_smc_axi_req_o`, split off by the route demux in `sep_system_peripherals` and wired to SMC's `sep_axi_in`. It reaches every SMC local target without passing the SEP outbound filter. SEP also has a local SPI host for boot-flash access. The SPI chapter describes that controller; SMC documents the shared peripherals.

SEP boot uses configuration straps, which it reads from the SMC strap registers. For their role in selecting boot options, see Configuration Straps and BL0 Boot Flow.

Figure 23 shows the connection from SEP to peripherals shared with SMC.

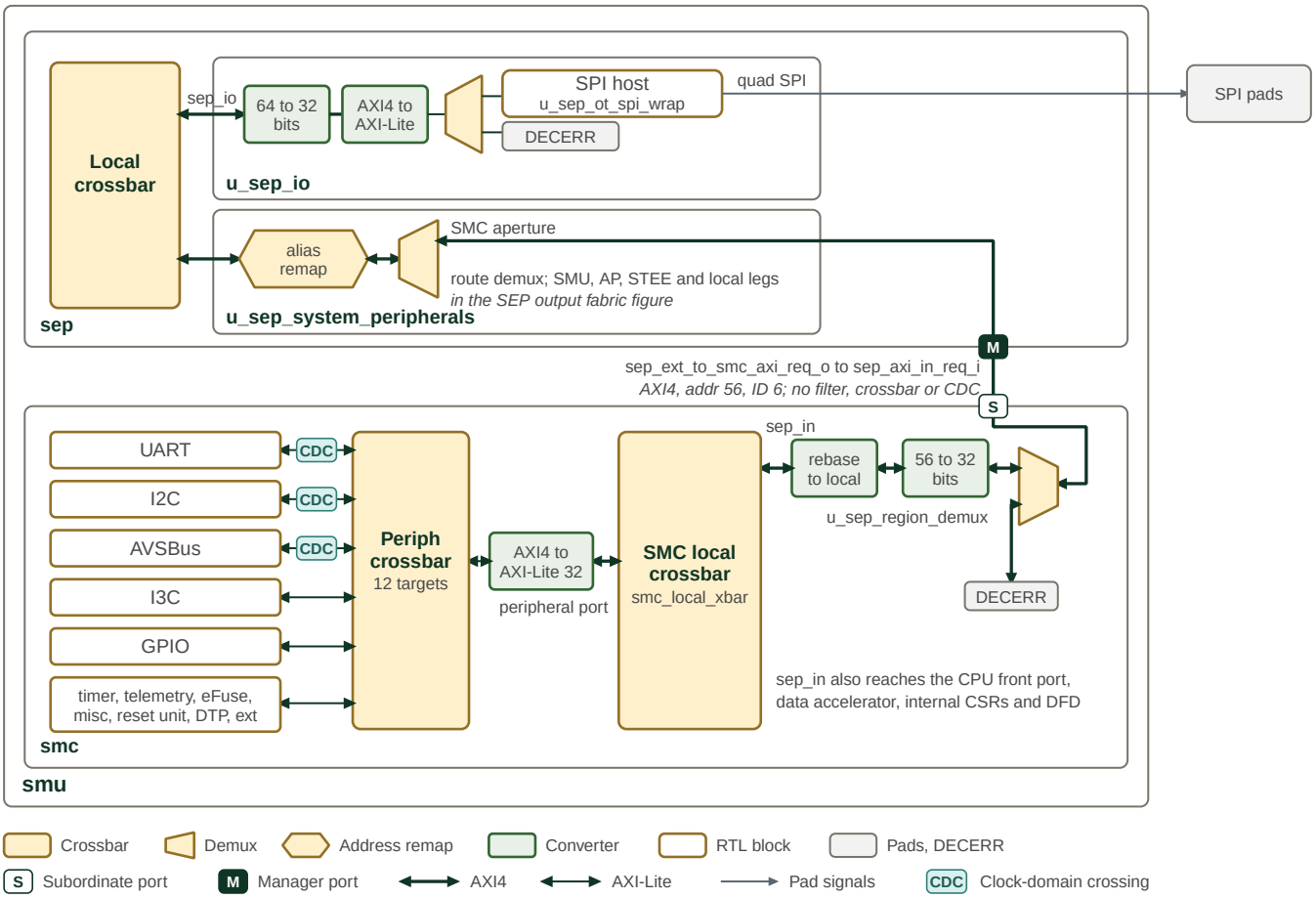


*Figure 23. SEP peripheral extension port*

The figure reads from the SEP local crossbar outwards. The SPI host sits behind the `sep_io` target, which downsizes the 64-bit AXI4 request to 32 bits and converts it to AXI-Lite; the quad SPI signals leave for the SPI pads. The local crossbar sends only the SPI host register extent to `sep_io` and answers the rest of its aperture with DECERR on reads and writes. The error responder in `sep_io` answers only the beats of a burst that run past the extent. Requests in the SMC aperture take the route demux of `sep_system_peripherals` (see Figure 18) onto the SMC port, which the SMU wires directly to the SMC's `sep_axi_in` port with no crossbar and no clock-domain crossing. The SMC flow is drawn right to left. The SMC input fabric passes addresses in the SMC local or global window, truncated to 32 bits, and answers others with DECERR. The SMC local fabric rebases them onto the SMC local base, and the `sep_in` initiator of the SMC local crossbar reaches every target, including the peripheral port, which is converted to 32-bit AXI-Lite for the SMC peripheral crossbar. The UART, I2C and AVSBus controllers sit behind clock-domain crossings into the peripheral clock.

### Serial Peripheral Interface (SPI)

SEP instantiates one OpenTitan SPI Host for software-controlled access to external devices. NOR flash is the primary source of the BL1 boot image. The controller uses commands and transmit/receive FIFOs; it does not provide a memory-mapped flash window for execute-in-place (XIP) access.

### Use Cases

- Read the BL1 boot image from external NOR flash.
- Connect to sensors, particularly in automotive applications.

The intended flash interface uses the x1 and x4 modes of MT25-class devices. The host cannot drive the octal x8 DDR interface of MT35-class devices. Select a flash device and command protocol compatible with the supported modes below.

### Features

*Table 96. Features*

| Capability | SEP implementation |
|---|---|
| Transfer width | Standard (x1), dual (x2) and quad (x4), selected per command segment. |
| Transfer rate | Single transfer rate only. No octal (x8), DDR or DQS read-data strobe. |
| Chip select | One chip-select output. The wrapper parameter `NUM_CS` supports other counts, but `sep_io` instantiates it with `NUM_CS = 1`. |
| Data buffering | Separate transmit and receive FIFOs, with independently defined depths in the core configuration. |
| Register interface | 32-bit AXI4-Lite converted to the core's native TL-UL interface. |
| External signals | SCK, active-low chip select and four bidirectional data lines, with output enables. |

The frequency targets are up to 80 MHz in x1 mode and 166 MHz in x4 mode. These are design targets, not guaranteed operating limits for an integrated platform. Software sets the serial clock through `CONFIGOPTS.CLKDIV`: `f_SCK = f_clk_i / (2 * (CLKDIV + 1))`. Select the divider and sampling timing for the attached device and the platform's validated clock and I/O timing.

### Integration and Register Access

`sep_io` narrows the 64-bit peripheral AXI bus to 32 bits. The `sep_ot_spi_wrap` wrapper then converts AXI4-Lite accesses to TL-UL for the vendored OpenTitan SPI Host. Software submits command segments and moves data through the FIFOs; controller activity is available in `STATUS.ACTIVE`, not on a separate busy output.

An 8-byte write is split into two 4-byte writes. If the byte strobes select only one half, the other reaches the AXI4-Lite bridge with all strobes clear. The bridge completes that no-op locally with OKAY and sends no TL-UL request. Forwarding it to the core would set `ERROR_STATUS.ACCESSINVAL` and return an error.

The core's access-control (RACL) interface is disabled and its SPI pass-through interface is tied inactive. SEP contains no SPI slave device and does not use the pass-through arrangement that lets a discrete secure element interpose on another SPI device's traffic. An integration requiring XIP, octal or DDR operation needs a different controller; `sep_io_spi` is the existing SPI integration boundary.

### Interrupts

The core's error and SPI-event interrupt outputs are ORed onto one line. The `irq` field of `sep_io_spi_req_t` carries that line to `sep_internal_interrupts` and the SEP PIC. Software reads `INTR_STATE` to identify the source.

*Table 97. Interrupts*

| `INTR_STATE` bit | Behavior and clearing |
|---|---|
| `ERROR` | Latched event. An enabled error event or an `INTR_TEST` write sets it. It remains set after the cause disappears or `INTR_TEST` is written back to zero. Clear it by writing 1 to the state bit; clearing `ERROR_STATUS` alone does not clear it. |
| `SPI_EVENT` | Level-sensitive status of the conditions selected by `EVENT_ENABLE`. Service the condition or change its selection to remove the source; it is not a latched error event. |

`INTR_ENABLE` qualifies only the outgoing interrupt line. It does not gate either state bit. `ERROR_ENABLE` selects the configurable error events; invalid register accesses always generate an error event.

### DMA Trigger

The host's `lsio_trigger_o` connects to Secure DMA handshake input 0. It is the registered OR of the transmit and receive watermark conditions: `TXQD < TX_WATERMARK` or `RXQD >= RX_WATERMARK`. These thresholds are in 32-bit FIFO words. The trigger does not depend on `EVENT_ENABLE` or `INTR_ENABLE`; interrupt selection and DMA pacing are separate controls. Configure the FIFO watermarks for the intended transfer direction and use the DMA chapter's handshake setup.

### Core Reference

The OpenTitan SPI Host documentation describes core operation and command sequencing. The register map below records the integrated core's registers and fields. The Programmer's Guide gives the programming guidance for this block.

### Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

### SPI Host

Serial peripheral interface for host mode, suitable for interfacing external serial NOR flash devices

*Table 98. SPI Host register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | CONTROL | RW | - |
| 0x14 | STATUS | R | - |
| 0x18 | CONFIGOPTS | RW | - |
| 0x1C | CSID | RW | - |
| 0x20 | COMMAND | W | - |
| 0x2C | ERROR_ENABLE | RW | - |
| 0x30 | ERROR_STATUS | RW | - |
| 0x34 | EVENT_ENABLE | RW | - |

### INTR_STATE

*Table 99. spi_controller.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `SPI_EVENT` | R | 0x0 | Event-related interrupts, see !!EVENT_ENABLE register for more information. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `ERROR` | RW1C | 0x0 | Error-related interrupts, see !!ERROR_ENABLE register for more information. |

### INTR_ENABLE

*Table 100. spi_controller.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `SPI_EVENT` | RW | 0x0 | Enable interrupt when !!INTR_STATE.spi_event is set. |
| 0 | `ERROR` | RW | 0x0 | Enable interrupt when !!INTR_STATE.error is set. |

### INTR_TEST

*Table 101. spi_controller.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `SPI_EVENT` | W | 0x0 | Write 1 to force !!INTR_STATE.spi_event to 1. |
| 0 | `ERROR` | W | 0x0 | Write 1 to force !!INTR_STATE.error to 1. |

### ALERT_TEST

*Table 102. spi_controller.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `FATAL_FAULT` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### CONTROL

*Table 103. spi_controller.CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `SPIEN` | RW | 0x0 | Enables the SPI host. On reset, this field is 0, meaning that no transactions can proceed. |
| 30 | `SW_RST` | RW | 0x0 | Clears the internal state (not registers) to the reset state when set to 1, including the FIFOs, the CDC's, the core state machine and the shift register. In the current implementation, the CDC FIFOs are drained not reset. Therefore software must confirm that both FIFO's empty before releasing the IP from reset. |
| 29 | `OUTPUT_EN` | RW | 0x0 | Enable the SPI host output buffers for the sck, csb, and sd lines. This allows the SPI_HOST IP to connect to the same bus as other SPI controllers without interference. |
| 28:16 | `Reserved` | - | - | Reserved |
| 15:8 | `TX_WATERMARK` | RW | 0x0 | If !!EVENT_ENABLE.TXWM is set, the IP will send an interrupt when the depth of the TX FIFO drops below TX_WATERMARK words (32b each). |
| 7:0 | `RX_WATERMARK` | RW | 0x7F | If !!EVENT_ENABLE.RXWM is set, the IP will send an interrupt when the depth of the RX FIFO reaches RX_WATERMARK words (32b each). |

### STATUS

*Table 104. spi_controller.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `READY` | R | 0x0 | When high, indicates the SPI host is ready to receive commands. Writing to COMMAND when READY is low is an error, and will trigger an interrupt. |
| 30 | `ACTIVE` | R | 0x0 | When high, indicates the SPI host is processing a previously issued command. |
| 29 | `TXFULL` | R | 0x0 | When high, indicates that the transmit data fifo is full. Any further writes to !!TXDATA will create an error interrupt. |
| 28 | `TXEMPTY` | R | 0x0 | When high, indicates that the transmit data fifo is empty. |
| 27 | `TXSTALL` | R | 0x0 | If high, signifies that an ongoing transaction has stalled due to lack of data in the TX FIFO |
| 26 | `TXWM` | R | 0x0 | If high, the amount of data in the TX FIFO has fallen below the level of !!CONTROL.TX_WATERMARK words (32b each). |
| 25 | `RXFULL` | R | 0x0 | When high, indicates that the receive fifo is full. Any ongoing transactions will stall until firmware reads some data from !!RXDATA. |
| 24 | `RXEMPTY` | R | 0x0 | When high, indicates that the receive fifo is empty. Any reads from RX FIFO will cause an error interrupt. |
| 23 | `RXSTALL` | R | 0x0 | If high, signifies that an ongoing transaction has stalled due to lack of available space in the RX FIFO |
| 22 | `BYTEORDER` | R | 0x0 | The value of the ByteOrder parameter, provided so that firmware can confirm proper IP configuration. |
| 21 | `Reserved` | - | - | Reserved |
| 20 | `RXWM` | R | 0x0 | If high, the number of 32-bits in the RX FIFO now exceeds the !!CONTROL.RX_WATERMARK entries (32b each). |
| 19:16 | `CMDQD` | R | 0x0 | Command queue depth. Indicates how many unread 32-bit words are currently in the command segment queue. |
| 15:8 | `RXQD` | R | 0x0 | Receive queue depth. Indicates how many unread 32-bit words are currently in the RX FIFO. When active, this result may an underestimate due to synchronization delays. |
| 7:0 | `TXQD` | R | 0x0 | Transmit queue depth. Indicates how many unsent 32-bit words are currently in the TX FIFO. When active, this result may be an overestimate due to synchronization delays. |

### CONFIGOPTS

*Table 105. spi_controller.CONFIGOPTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `CPOL` | RW | 0x0 | The polarity of the sck clock signal. When CPOL is 0, sck is low when idle, and emits high pulses. When CPOL is 1, sck is high when idle, and emits a series of low pulses. |
| 30 | `CPHA` | RW | 0x0 | The phase of the sck clock signal relative to the data. When CPHA = 0, the data changes on the trailing edge of sck and is typically sampled on the leading edge. Conversely if CPHA = 1 high, data lines change on the leading edge of sck and are typically sampled on the trailing edge. CPHA should be chosen to match the phase of the selected device. The sampling behavior is modified by the !!CONFIGOPTS.FULLCYC bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 29 | `FULLCYC` | RW | 0x0 | Full cycle. Modifies the CPHA sampling behaviour to allow for longer device logic setup times. Rather than sampling the SD bus a half cycle after shifting out data, the data is sampled a full cycle after shifting data out. This means that if CPHA = 0, data is shifted out on the trailing edge, and sampled a full cycle later. If CPHA = 1, data is shifted and sampled with the trailing edge, also separated by a full cycle. |
| 28 | `Reserved` | - | - | Reserved |
| 27:24 | `CSNLEAD` | RW | 0x0 | CS_N Leading Time. Indicates the number of half sck cycles, CSNLEAD+1, to leave between the falling edge of cs_n and the first edge of sck. Setting this register to zero corresponds to the minimum delay of one-half sck cycle |
| 23:20 | `CSNTRAIL` | RW | 0x0 | CS_N Trailing Time. Indicates the number of half sck cycles, CSNTRAIL+1, to leave between last edge of sck and the rising edge of cs_n. Setting this register to zero corresponds to the minimum delay of one-half sck cycle. |
| 19:16 | `CSNIDLE` | RW | 0x0 | Minimum idle time between commands. Indicates the minimum number of sck half-cycles to hold cs_n high between commands. Setting this register to zero creates a minimally-wide CS_N-high pulse of one-half sck cycle. |
| 15:0 | `CLKDIV` | RW | 0x0 | Core clock divider. Slows down subsequent SPI transactions by a factor of (CLKDIV+1) relative to the core clock frequency. The period of sck, T(sck) then becomes `2*(CLK_DIV+1)*T(core)` |

### CSID

*Table 106. spi_controller.CSID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `CSID` | RW | 0x0 | Chip Select ID |

### COMMAND

*Table 107. spi_controller.COMMAND fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | `Reserved` | - | - | Reserved |
| 24:5 | `LEN` | W | 0x0 | Segment Length.<br><br>For read or write segments, this field controls the number of 1-byte bursts to transmit and or receive in this command segment. The number of cycles required to send or received a byte will depend on !!COMMAND.SPEED. For dummy segments, (!!COMMAND.DIRECTION == 0), this register controls the number of dummy cycles to issue. The number of bytes (or dummy cycles) in the segment will be equal to !!COMMAND.LEN + 1. |
| 4:3 | `DIRECTION` | W | 0x0 | The direction for the following command: "0" = Dummy cycles (no TX/RX). "1" = Rx only, "2" = Tx only, "3" = Bidirectional Tx/Rx (Standard SPI mode only). |
| 2:1 | `SPEED` | W | 0x0 | The speed for this command segment: "0" = Standard SPI. "1" = Dual SPI. "2"=Quad SPI, "3": RESERVED. |
| 0 | `CSAAT` | W | 0x0 | **C**hip **S**elect **A**ctive **A**fter **T**ransaction. If !!COMMAND.CSAAT = 0, the chip select line is raised immediately at the end of the command segment. If !!COMMAND.CSAAT = 1, the chip select line is left low at the end of the current transaction segment. This allows the creation of longer, more complete SPI transactions, consisting of several separate segments for issuing instructions, pausing for dummy cycles, and transmitting or receiving data from the device. |

### ERROR_ENABLE

*Table 108. spi_controller.ERROR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `CSIDINVAL` | RW | 0x1 | Invalid CSID: If this bit is set, the block sends an error interrupt whenever a command is submitted, but CSID exceeds NumCS. |
| 3 | `CMDINVAL` | RW | 0x1 | Invalid Command Errors: If this bit is set, the block sends an error interrupt whenever a command is sent with invalid values for !!COMMAND.SPEED or !!COMMAND.DIRECTION. |
| 2 | `UNDERFLOW` | RW | 0x1 | Underflow Errors: If this bit is set, the block sends an error interrupt whenever there is a read from !!RXDATA but the RX FIFO is empty. |
| 1 | `OVERFLOW` | RW | 0x1 | Overflow Errors: If this bit is set, the block sends an error interrupt whenever the TX FIFO overflows. |
| 0 | `CMDBUSY` | RW | 0x1 | Command Error: If this bit is set, the block sends an error interrupt whenever a command is issued while busy (i.e. a 1 is when !!STATUS.READY is not asserted.) |

### ERROR_STATUS

*Table 109. spi_controller.ERROR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `ACCESSINVAL` | RW1C | 0x0 | Indicates that TLUL attempted to write to TXDATA with no bytes enabled. Such 'zero byte' writes are not supported. |
| 4 | `CSIDINVAL` | RW1C | 0x0 | Indicates a command was attempted with an invalid value for !!CSID. |
| 3 | `CMDINVAL` | RW1C | 0x0 | Indicates an invalid command segment, meaning either an invalid value of !!COMMAND.SPEED or a request for bidirectional data transfer at dual or quad speed |
| 2 | `UNDERFLOW` | RW1C | 0x0 | Indicates that firmware has attempted to read from !!RXDATA when the RX FIFO is empty. |
| 1 | `OVERFLOW` | RW1C | 0x0 | Indicates that firmware has overflowed the TX FIFO |
| 0 | `CMDBUSY` | RW1C | 0x0 | Indicates a write to !!COMMAND when !!STATUS.READY = 0. |

### EVENT_ENABLE

*Table 110. spi_controller.EVENT_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `IDLE` | RW | 0x0 | Assert to send a spi_event interrupt whenever !!STATUS.ACTIVE goes low |
| 4 | `READY` | RW | 0x0 | Assert to send a spi_event interrupt whenever !!STATUS.READY goes high |
| 3 | `TXWM` | RW | 0x0 | Assert to send a spi_event interrupt whenever the number of 32-bit words in the TX FIFO is less than !!CONTROL.TX_WATERMARK. To prevent the reassertion of this interrupt add more data to the TX FIFO, or reduce !!CONTROL.TX_WATERMARK. |
| 2 | `RXWM` | RW | 0x0 | Assert to send a spi_event interrupt whenever the number of 32-bit words in the RX FIFO is greater than !!CONTROL.RX_WATERMARK. To prevent the reassertion of this interrupt, read more data from the RX FIFO, or increase !!CONTROL.RX_WATERMARK. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `TXEMPTY` | RW | 0x0 | Assert to send a spi_event interrupt whenever !!STATUS.TXEMPTY goes high |
| 0 | `RXFULL` | RW | 0x0 | Assert to send a spi_event interrupt whenever !!STATUS.RXFULL goes high |

### Watchdog Timer (WDT)

The Watchdog timer is a component used to enhance reliability and ensure that the system is notified if the SEP subsystem is ``hanging''. The SEP CPU must periodically visit the WDT and reset its counter, proving that the CPU is still alive and well i.e. not hanging. If the WDT timer counts up to its bark threshold it fires an external interrupt, and if it counts up to its bite threshold it requests a reset, both indicating a timeout has expired and there is a problem within the SEP.

### Bus Interface

The `sep_wdt_wrap` module integrates the OpenTitan `aon_timer` IP as the watchdog timer. The 64-bit AXI slave port from the system crossbar is narrowed to 32-bit AXI through an `axi_dw_converter`. The address path subtracts the WDT base address so that the internal TL-UL decoder sees zero-based offsets. After `axi_to_axi_lite` and `axi_lite_to_tlul` conversion, the TL-UL bus drives the standard `aon_timer` register block.

### Clock, Reset, and Debug Behavior

The wrapper provides a dedicated watchdog clock input (`clk_wdt_i`) that is synchronized through a `prim_sync_reset` to produce the `rst_aon_ni` reset for the always-on timer domain. The `wdt_debug_sleep_mode_i` input controls whether the timer continues to count during debug halt. Life-cycle escalation is tied to its inactive state. Alert receivers are initialized to the idle encoding to avoid spurious integrity errors; alert outputs are left unconnected since the SEP aggregates watchdog events through the interrupt path.

### Bark Interrupt and Bite Reset

The wrapper exports two signals: `intr_wdog_timer_bark_o`, the watchdog bark interrupt that fires when the first timeout stage expires, and `wdt_timer_rst_req_o`, the bite reset request that fires when the second timeout stage expires and the system failed to respond to the bark.

### Programming Reference

Consult the OpenTitan Always-On Timer documentation for the authoritative core functionality of the IP. The Programmer's Guide gives the programming guidance for this block. The register map follows below.

### Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

### Always-On Timer

Wakeup and watchdog timers running on a low-power, always-on clock

*Table 111. Always-On Timer register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | ALERT_TEST | W | - |
| 0x4 | WKUP_CTRL | RW | - |
| 0x8 | WKUP_THOLD_HI | RW | - |
| 0xC | WKUP_THOLD_LO | RW | - |
| 0x10 | WKUP_COUNT_HI | RW | - |
| 0x14 | WKUP_COUNT_LO | RW | - |
| 0x18 | WDOG_REGWEN | RW | - |
| 0x1C | WDOG_CTRL | RW | - |
| 0x20 | WDOG_BARK_THOLD | RW | - |
| 0x24 | WDOG_BITE_THOLD | RW | - |
| 0x28 | WDOG_COUNT | RW | - |
| 0x2C | INTR_STATE | RW | - |
| 0x30 | INTR_TEST | W | - |
| 0x34 | WKUP_CAUSE | RW | - |

### ALERT_TEST

*Table 112. aon_timer.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `fatal_fault` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### WKUP_CTRL

*Table 113. aon_timer.WKUP_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12:1 | `prescaler` | RW | 0x0 | Pre-scaler value for wakeup timer count |
| 0 | `enable` | RW | 0x0 | When set to 1, the wakeup timer will count |

### WKUP_THOLD_HI

*Table 114. aon_timer.WKUP_THOLD_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `threshold_hi` | RW | 0x0 | The count at which a wakeup interrupt should be generated, top 32 bits. |

### WKUP_THOLD_LO

*Table 115. aon_timer.WKUP_THOLD_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `threshold_lo` | RW | 0x0 | The count at which a wakeup interrupt should be generated, bottom 32 bits. |

### WKUP_COUNT_HI

*Table 116. aon_timer.WKUP_COUNT_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `count_hi` | RW | 0x0 | The current wakeup counter value, top 32 bits. |

### WKUP_COUNT_LO

*Table 117. aon_timer.WKUP_COUNT_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `count_lo` | RW | 0x0 | The current wakeup counter value, bottom 32 bits. |

### WDOG_REGWEN

*Table 118. aon_timer.WDOG_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `regwen` | RW0C | 0x1 | Once cleared, the watchdog configuration will be locked until the next reset. |

### WDOG_CTRL

*Table 119. aon_timer.WDOG_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `pause_in_sleep` | RW | 0x0 | When set to 1, the watchdog timer will not count during sleep. |
| 0 | `enable` | RW | 0x0 | When set to 1, the watchdog timer will count. |

### WDOG_BARK_THOLD

*Table 120. aon_timer.WDOG_BARK_THOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `threshold` | RW | 0x0 | The count at which a watchdog bark interrupt should be generated. |

### WDOG_BITE_THOLD

*Table 121. aon_timer.WDOG_BITE_THOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `threshold` | RW | 0x0 | The count at which a watchdog bite reset should be generated. |

### WDOG_COUNT

*Table 122. aon_timer.WDOG_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `count` | RW | 0x0 | The current watchdog counter value. |

#### INTR_STATE

*Table 123. aon_timer.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `wdog_timer_bark` | RW1C | 0x0 | Raised if the watchdog timer has hit the bark threshold. |
| 0 | `wkup_timer_expired` | RW1C | 0x0 | Raised if the wakeup timer has hit the specified threshold. |

#### INTR_TEST

*Table 124. aon_timer.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `wdog_timer_bark` | W | - | Write 1 to force wdog_timer_bark interrupt. |
| 0 | `wkup_timer_expired` | W | - | Write 1 to force wkup_timer_expired interrupt. |

#### WKUP_CAUSE

*Table 125. aon_timer.WKUP_CAUSE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `cause` | RW0C | 0x0 | AON timer requested wakeup, write 0 to clear. |

### 2.1.5.10. Security and Lifecycle

These blocks establish the chiplet's lifecycle state, process authorization tokens and control access to security assets, debug and test facilities. The Threat Model and Attack Countermeasures chapters describe the security objectives and protections.

#### OTP Fuse Controller

The One-Time Programmable (OTP) fuse controller stores persistent chiplet policy. At cold reset it senses the fuse bank into shadow registers for software access and hardware policy enforcement. Programming a fuse and updating its shadow are separate operations.

Figure 24 shows the fuse-bank organization.

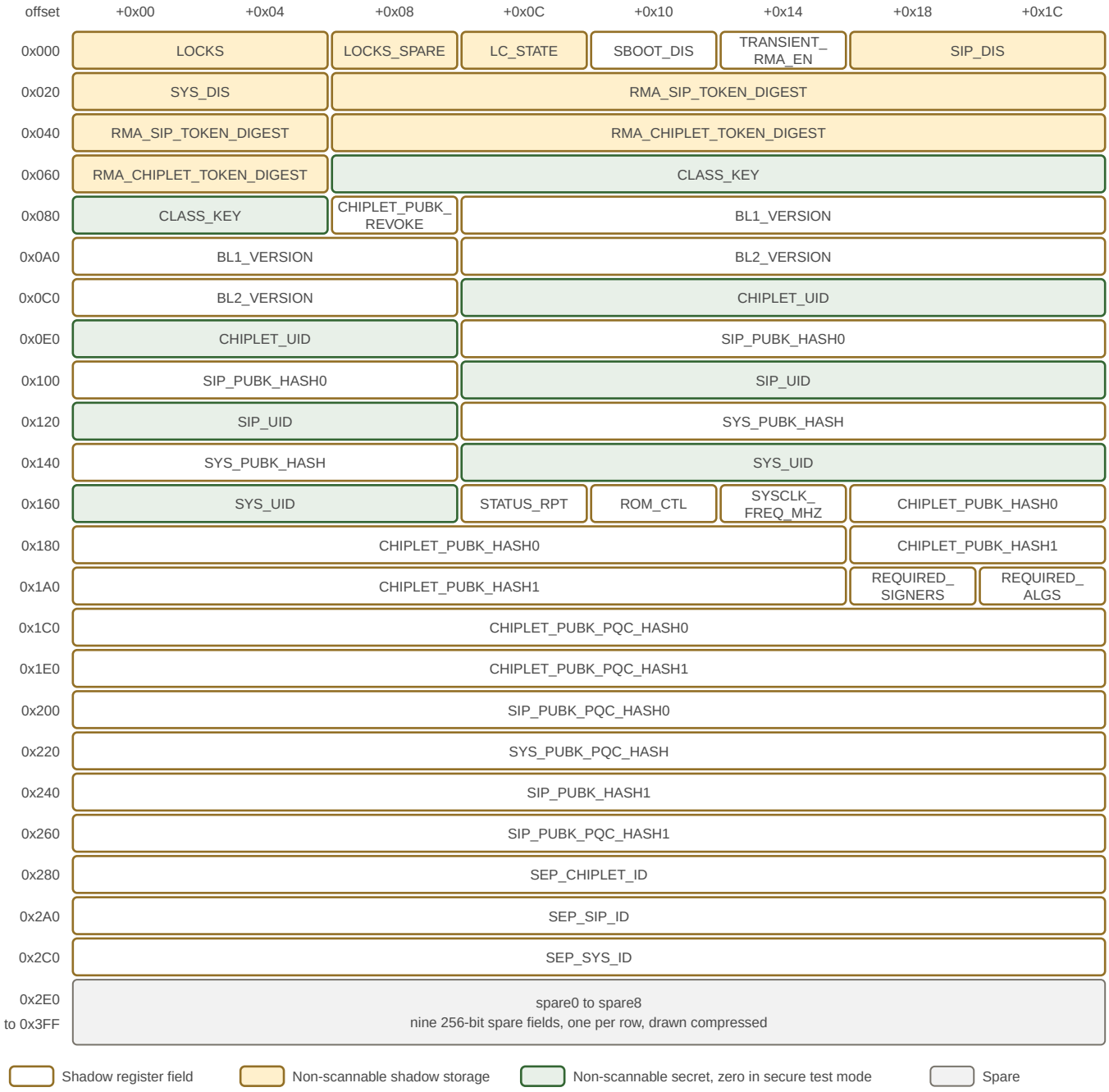


*Figure 24. SEP OTP fuse bank*

Each row of Figure 24 is 32 bytes, eight 32-bit words, at the offset given on the left from the base of the MAP window; a field that crosses a row boundary continues at the start of the next row. The shading gives the storage class of each field's shadow register. The lock field, the lifecycle state, the two disable vectors and the two RMA token digests are held in non-scannable storage. CLASS_KEY, CHIPLET_UID, SIP_UID and SYS_UID are non-scannable as well and read as zero downstream while SECURE_TM is 1. The nine spare fields fill the last nine rows and are drawn as one band.

Figure 25 shows the controller boundary and the technology-specific fuse interface.

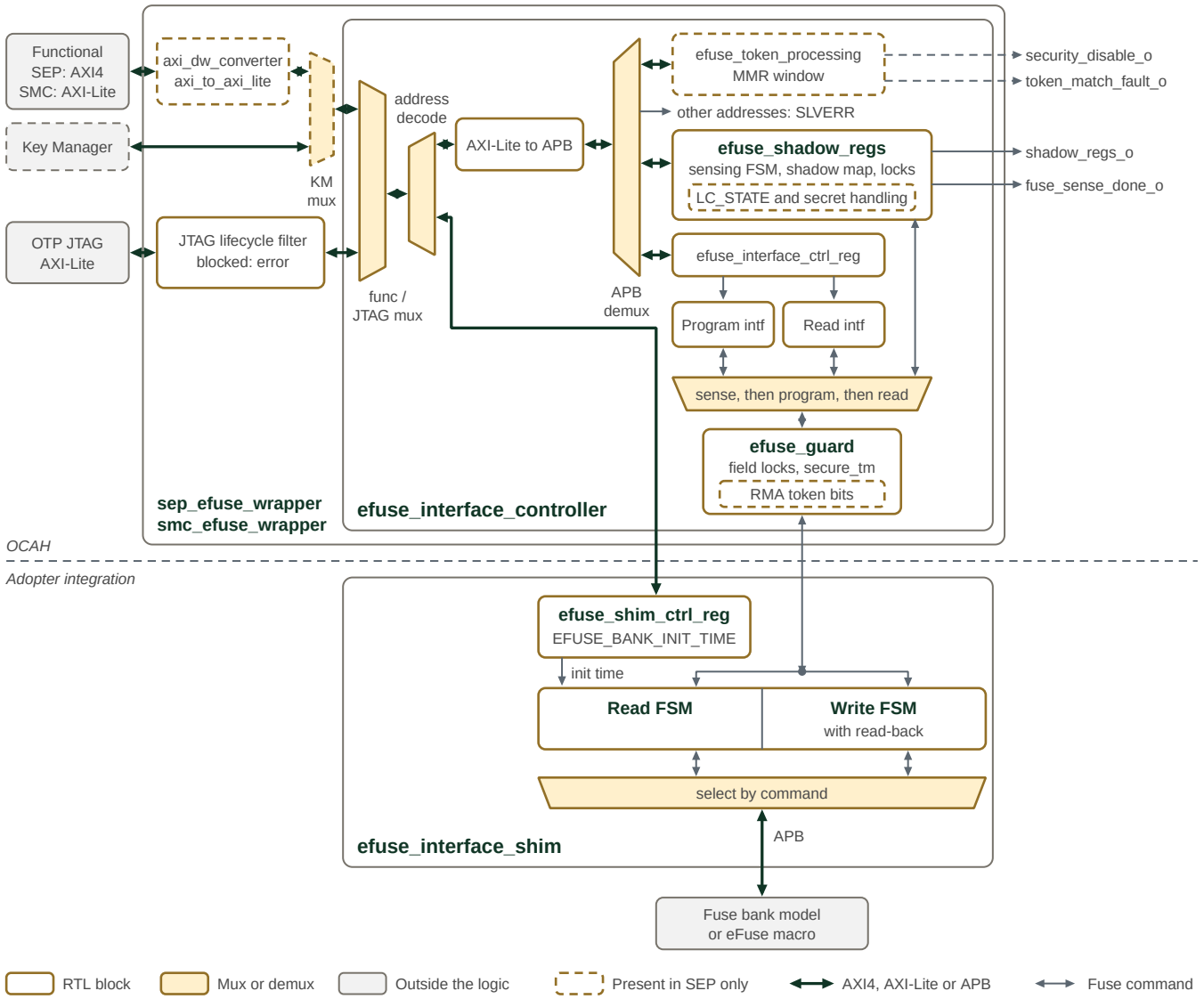


*Figure 25. eFuse controller and technology-specific interface shim*

The figure reads from the bus ports on the left to the fuse bank at the bottom. Thick arrows are AXI4, AXI-Lite or APB, and thin double-headed arrows carry a fuse command and its response. The dashed line separates the OCAH controller from the interface shim, which an adopter replaces with one for its fuse technology.

Both wrappers pass the OTP JTAG port through a lifecycle filter that answers a blocked access with an error response. Inside the controller, an AXI-Lite multiplexer merges the functional and JTAG ports. Addresses from the MAP window to the end of the last register window go through the AXI-Lite to APB bridge, and all others go to the shim's bank control registers. The APB decode answers any address in that span outside the register windows with SLVERR. The command multiplexer gives automatic sensing priority until it completes, then gives programming priority over reading, and `efuse_guard` filters every command before it reaches the shim.

The SMC reuses the controller in `smc_efuse_wrapper` with `HAS_LC_STATE=0`, so the logic with a dashed outline is absent there:

- The MMR window and `efuse_token_processing`: the SMC address map has no MMR window, and `security_disable_o` and `token_match_fault_o` are tied low.
- The LC_STATE handling in `efuse_shadow_regs`: the held INVALID output before sensing, the token-gated LC_STATE writes and the secret masking in secure test mode. The SMC has no secret-shadow ranges and ties `secure_tm_i` low.

- The RMA token-bit gate in `efuse_guard`.
- The width converter and the Key Manager port of `sep_efuse_wrapper`: the SMC functional port is a 32-bit AXI-Lite port connected to the controller directly.

The SMC has no local lifecycle controller. Its JTAG lifecycle filter decodes the lifecycle state received from SEP, and its `security_disable_i` input takes the SEP security-disable status, which skips fuse sensing and opens the shadow registers to software.

### Fuse Sensing and Boot Sequencing

When SEP is released from reset (chiplet_cold_reset), the sensing FSM automatically copies each fuse field into its dedicated memory-mapped shadow register.

The logical `LC_STATE` output remains `4'b1111` until fuse sensing completes; only then is the LC_STATE value sensed from the fuses driven to downstream logic such as the lifecycle controller. Table 126 defines whether software can read and write the shadow register of each fuse field. Shadow registers that are not software-visible drive their fuse values directly to downstream hardware. All fuse fields other than LOCKS and LC_STATE are made available to downstream logic as soon as they are ready.

After the fuses have been read into their shadow registers, a sense-completion indicator is generated. This triggers the DFT-based memory-repair machine for the SEP SRAMs. Once memory repair completes or is bypassed, the SEP CPU is enabled and fetches its first instruction from the SEP ROM.

Figure 26 shows the fuse-sensing sequence and shadow-register timing.

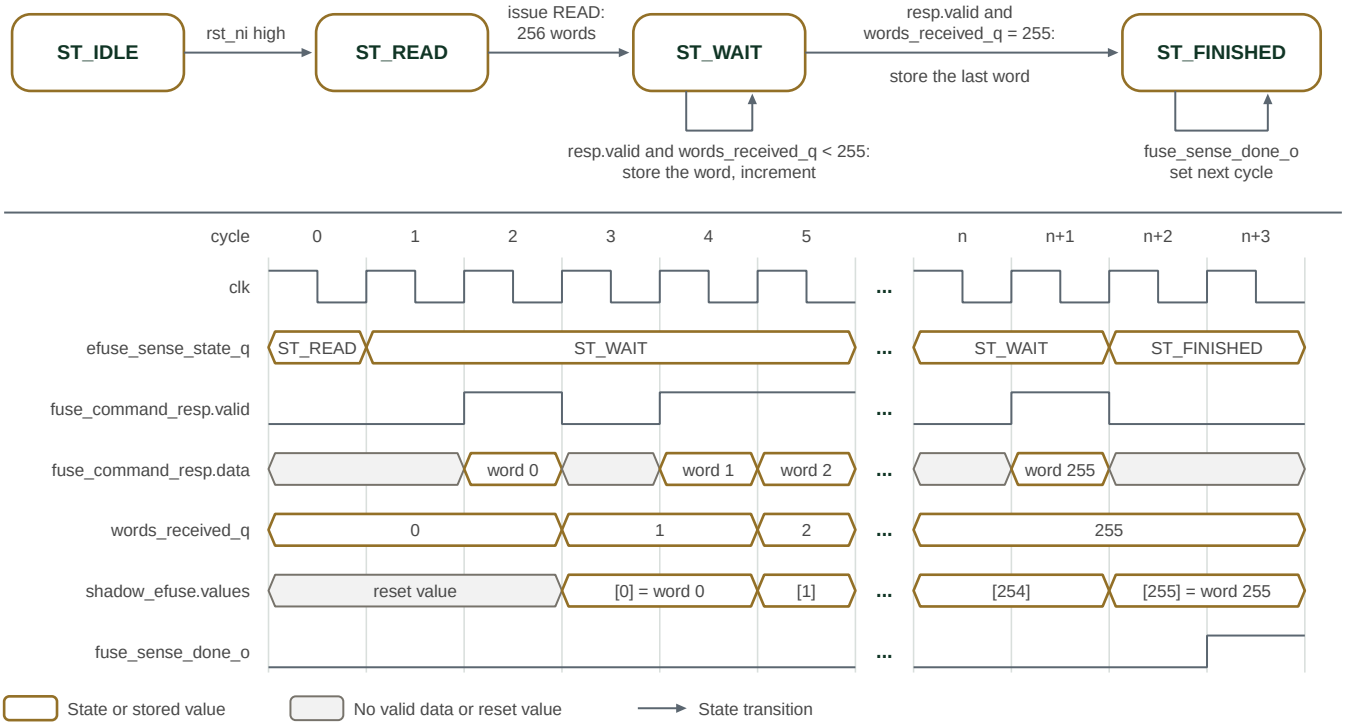


*Figure 26. eFuse sensing state machine and shadow-register timing*

The upper part of Figure 26 shows the four states of the sensing FSM in `efuse_shadow_regs`, and the lower part shows one sensing run cycle by cycle. ST_READ issues a single READ command for all 256 shadow words. The shim then returns one `fuse_command_resp.valid` strobe per word, and strobes may arrive on consecutive cycles. Each strobe stores its data

in the shadow word indexed by `words_received_q` and increments the counter; a strobe with its status bit set still advances the counter but leaves that word unwritten. The strobe that arrives while the counter is 255 stores the last word and moves the FSM to ST_FINISHED, and `fuse_sense_done_o` rises one cycle later. Until then, APB accesses to the MAP window return SLVERR.

#### Shadow Register and JTAG Access Control

Each assigned lock slot contains an independent write-lock and read-lock bit. A write-lock blocks fuse programming and shadow writes; a read-lock blocks software and JTAG reads of both the fuse field and its shadow. The sensing FSM can still read locked fuses to populate the shadows.

There are two exceptions: the `LOCKS` and `LOCKS_SPARE` registers have no lock slot of their own, and `LC_STATE` ignores its assigned lock bits. Lifecycle transition and token checks still apply to `LC_STATE`. The slot numbers and bit positions are defined in Table 126.

The fuse block's internal space, including the fuse-programming configuration space, is accessible through both a JTAG interface and an AXI interface, as shown in the figure, and each interface has its own access control. In addition to the shadow-register access control defined by Table 126, the JTAG port shall be further restricted: in the PROD and RMA_SIP states it may reach only the eFuse MMR token block. That block is the three token-input arrays (RMA_SIP_TOKEN_I, RMA_CHIPLET_TOKEN_I and SEC_DISABLE_TOKEN_I), the TOKEN_EOP register that triggers a match, and the three match-status registers. Every other JTAG access to the fuse block — the shadow registers, the interface control registers and the SHIM — shall complete with an error response.

#### Fuse Programming and RMA Authorization

The fuse-programming protection shall enable the fuse-programming function only after the LOCKS field has been sensed, and shall honor the write-lock value of each field. Programming LC_STATE is subject to additional restrictions: LC_STATE[2:1] indicate whether the chiplet is in an RMA state (see the LC State Machine section of the Life Cycle Controller). The SEP fuse controller implements two 256-bit registers, RMA_SIP_TOKEN_I and RMA_CHIPLET_TOKEN_I.

These two registers can be populated by both SEP software and JTAG. Each input is hashed (SHA-256) and compared with the stored RMA_SIP_TOKEN_DIGEST and RMA_CHIPLET_TOKEN_DIGEST, respectively. Only on a match can LC_STATE[1] and LC_STATE[2] be programmed; the token matching section of the Programmer's Guide gives the firmware sequence. For the matching mechanism, see Token Matching Mechanism and Attack Countermeasures.

#### Programming Voltage

The fuse bank shall support keeping the programming-voltage rail active for an unlimited lifetime. This is achieved by using only the internal charge-pump-generated programming voltage rather than the external programming voltage.

The charge pump is calibrated early in wafer test, and its calibration values are burned into the corresponding fuse bits; this is the only time the external fuse-programming voltage is used. Consequently, software-initiated fuse programming never needs to configure the programming voltage.

#### Shadow Register Updates and Test-Mode Protection

- Programming a fuse does not update its shadow register until the next fuse sensing occurs.
- The shadow registers of LOCK (both the LOCKS and LOCKS_SPARE windows), LC_STATE, SIP_DIS, and SYS_DIS are non-scannable. Any register on the path from these registers to the feature control outputs or the fuse write-enable signals is likewise non-scannable.
- When SECURE_TM is 1 (see the Test Mode section), the shadow registers of LOCK (both the LOCKS and LOCKS_SPARE windows), LC_STATE, SIP_DIS, and SYS_DIS shall be blocked from any update, from every

source — software, JTAG, and TRoot. This list is normative and exhaustive: these are the fields whose perturbation could enable a feature that a DIS vector or the lifecycle state has disabled. An implementation may block further fields, but no other requirement in this specification depends on it.

To prevent test patterns from accidentally or maliciously changing or reading out the fuse values:

- When SECURE_TM is 1, the fuse write-enable inputs to the fuse IP block are tied inactive, and the fuse-bank outputs are disconnected from downstream logic. If needed, dedicated test registers can be added to drive the downstream logic during test.

### Fuse Fields

Table 126 lists lock slots, field sizes, shadow access permissions and purposes. Slot numbers select lock bits; they are neither OTP word offsets nor register addresses. Access permissions remain subject to the lock and lifecycle rules above.

*Table 126. SEP fuse fields*

| Field | Lock slot | Size (bits) | SW writable | SW readable | Description |
|---|---|---|---|---|---|
| LOCK | — | 96 | set-only | true | Lock state occupies OTP words 0–2:<br><br>`LOCKS` (64 bits at `0x0`) covers slots 0–31: write-lock at bit `2n`, read-lock at bit `2n+1` for slot `n`.<br><br>`LOCKS_SPARE` (32 bits at `0x8`) covers slots 32–40: write-lock at bit `2(n-32)`, read-lock at bit `2(n-32)+1`. Bits [31:18] are unassigned slots 41–47.<br><br>Firmware reads the complete state as three consecutive 32-bit words. There are 41 assigned slots (0–40); slot 0 belongs to `LC_STATE`, which ignores both lock bits. LOCK has no lock-bit pair and cannot lock itself. |
| LC_STATE | 0 | 32 (4-bit state, 8-bit encoding) | set-only | true | Life-cycle state of the chiplet. The shadow encodes the logical state as `{~raw[3:0], raw[3:0]}` in bits [7:0]; the bit numbers below refer to the logical state. ***LC_STATE shall never be write-locked or read-locked by the corresponding LOCKS bits***. LC_STATE[3] and LC_STATE[0] are set-only. LC_STATE[2:1] shall be set-only when the corresponding *_TOKEN has a match. Setting bit 2 also requires bit 1 to be set in the pre-write state; one write cannot establish both RMA stages. Invalid lifecycle encodings are terminal. Otherwise, LC_STATE[2:1] cannot be updated. |
| SBOOT_DIS | 1 | 32 (use only 1 bit) | true | true | Disables secure boot when set to 1, in any lifecycle state. It overrides lifecycle enforcement in PROD and PROD_END and the TEST_DEV / RMA_SiP debug-lock rule, but not a signed manifest's request for verification; see Firmware Authorization in the Lifecycle Controller section. |
| TRANSIENT_RMA_EN | 2 | 32 (use only 1 bit) | true | true | Enables automatic RMA shadow-state updates after token matching, without requiring a reset. It does not enable the token matcher itself. The chiplet-token path has priority and requires an already established RMA_SIP state; a chiplet match without that state blocks the transition rather than falling through to a simultaneous SiP match. When 0, token matches do not automatically update `LC_STATE`; explicit shadow writes remain subject to token and transition checks. |
| SIP_DIS | 3 | 64 | set-only | true | SiP owner's feature-control disable vector, provisioned by the chiplet owner. A bit set to 1 disables the corresponding feature. |
| SYS_DIS | 4 | 64 | set-only | true | System owner's feature-control disable vector, provisioned by the SiP owner. A bit set to 1 disables the corresponding feature. |
| RMA_SIP_TOKEN_DIGEST | 5 | 256 | true | true | Hash digest of the secret RMA_SIP_TOKEN, that enables entering RMA_SIP Life Cycle state |

| Field | Lock slot | Size (bits) | SW writable | SW readable | Description |
|---|---|---|---|---|---|
| RMA_CHIPLET_TOKEN_DIGEST | 6 | 256 | true | true | Hash digest of the secret RMA_CHIPLET_TOKEN that enables entering RMA_CHIPLET Life Cycle state |
| CLASS_KEY | 7 | 256 | true | true | Can be used for firmware encryption and decryption. |
| CHIPLET_PUBK_REVOKE | 8 | 32 | set-only | true | Global public-key revocation vector covering every boot-verification key of every owner (classical and PQC). SEP ROM embeds a set of public-key hash digests and accesses additional fused key hashes; each key entry maps to one bit here. Once set, SEP ROM rejects the corresponding key. Bit map: [7:0] ROM chiplet-creator classical keys (bits 0,1 are development keys); [15:8] ROM chiplet-creator PQC keys; [16]/[17] CHIPLET_PUBK_HASH0/1; [18]/[19] CHIPLET_PUBK_PQC_HASH0/1; [20]/[21] SIP_PUBK_HASH0 / SIP_PUBK_PQC_HASH0; [22]/[23] SYS_PUBK_HASH / SYS_PUBK_PQC_HASH; [24]/[25] SIP_PUBK_HASH1 / SIP_PUBK_PQC_HASH1; [31:26] reserved. Before the chiplet owner transfers a part to the customer, the development-key bits shall be set (revoked). (Name retained for compatibility; scope is now all owners.) |
| BL1_VERSION | 9 | 256 | set-only | true | BL1 anti-rollback storage. The production ROM supplies the low 128 raw bits as security-version flags to the manifest validator; the upper 128 bits do not participate in that check. See Anti-Rollback Security-Version Flags. |
| BL2_VERSION | 10 | 256 | set-only | true | BL2 anti-rollback storage for later-stage firmware policy. This bank is not the production ROM's BL1 security-version input; software using it must define its encoding and enforcement policy. |
| CHIPLET_UID | 11 | 256 | true | true | Silicon creator's *secret* device unique ID (also referred to as the UDS, Unique Device Secret). The chiplet creator is responsible for CHIPLET_UID provisioning and public identity enrollment. Write-lock and read-lock independently control modification and software visibility. Hardware recording of provisioning measurements is a proposed enhancement. |
| SIP_PUBK_HASH0 | 12 | 256 | true | true | Hash digest of SiP owner's root public key for secure boot (slot 0). Note: The chiplet creator's actual root public key is embedded in the SEPROM. The digest is used to validate that public key. |
| SIP_UID | 13 | 256 | true | true | SiP owner's *secret* device unique ID. The SiP owner is responsible for SIP_UID provisioning and public identity enrollment. Write-lock and read-lock independently control modification and software visibility. Hardware recording of provisioning measurements and retention of prior-owner key/UID pairs in the derivation hierarchy are proposed enhancements. |
| SYS_PUBK_HASH | 14 | 256 | true | true | Hash digest of system owner's root public key for secure boot. |
| SYS_UID | 15 | 256 | true | true | System owner's *secret* device unique ID. The system owner is responsible for SYS_UID provisioning and public identity enrollment. Write-lock and read-lock independently control modification and software visibility. |
| STATUS_RPT | 16 | 32 (2 used) | true | true | When STATUS_RPT[1:0] == 2'b00, the boot ROM samples an SMC GPIO (STATUS_RPT) to determine whether to report status. When STATUS_RPT[1:0] == 2'b01, the boot ROM reports status; otherwise, it does not. |
| ROM_CTL | 17 | 32 | true | true | ROM_CTL[0]: 0 = little-endian, 1 = big-endian. ROM_CTL[5:1]: row-swap control; swaps ROM slices when set. ROM_CTL[31:6]: reserved. |
| SYSCLK_FREQ_MHZ | 18 | 32 (11 used) | true | true | Configured sysclk PLL frequency in MHz (`sysclk_freq_mhz[10:0]`). Ignored when 0. Consumed by boot ROM clock math (PLL selection and SPI `CLKDIV` derivation). |
| CHIPLET_PUBK_HASH0 | 19 | 256 | true | true | Hash of chiplet-owner authentication key 0, fused. These fused chiplet-owner key hashes can be used in addition to the hashes embedded in the ROM to authenticate manifests. |

| Field | Lock slot | Size (bits) | SW writable | SW readable | Description |
|---|---|---|---|---|---|
| CHIPLET_PUBK_HASH1 | 20 | 256 | true | true | Hash of chiplet-owner authentication key 1, fused. |
| REQUIRED_SIGNERS | 21 | 32 (use only 2 bits) | true | true | Owners whose signature a manifest must carry to be authenticated by SEP ROM. Per owner, 1 = required, 0 = not-required; the default all-zero value requires no additional owner. [0] SiP owner, [1] system owner, [31:2] reserved. The chiplet creator is the root of trust — its classical and PQC root digests are embedded in SEP ROM and always authenticate BL1 — so it has no bit here. Burning a fuse bit only ever adds a required signer, so the fuse is monotonic: the standard LOCK bits exist but locking this field is unnecessary, since a later burn can never weaken security. The value is applied via the shadow register (loaded from the fuse at cold reset, SW-writable): firmware may update the shadow to set the required signers for each manifest it verifies (BL1, BL2, runtime, …). See Secure-Boot Signing Model in the Life Cycle Controller section. |
| REQUIRED_ALGS | 22 | 32 (use only 12 bits) | set-only | true | Per-owner required signature algorithms, applied only when that owner is required. Each owner has a 4-bit nibble; bit 0 = classical (ECDSA P-384 or RSA), bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. The specific classical scheme and ML-DSA parameter set (e.g., ML-DSA-44/65/87) are set by the owner's provisioned keys, not mandated here. [3:0] chiplet, [7:4] SiP owner, [11:8] system owner, [31:12] reserved. For each required (owner, algorithm) the manifest must carry a signature verified against that owner's burned, non-revoked key of that type; all required pairs must verify, and any signature from a non-required owner is ignored. |
| CHIPLET_PUBK_PQC_HASH0 | 23 | 256 | true | true | Chiplet owner ML-DSA PQC authentication key 0 hash, fused. PQC counterpart of CHIPLET_PUBK_HASH0. |
| CHIPLET_PUBK_PQC_HASH1 | 24 | 256 | true | true | Chiplet owner ML-DSA PQC authentication key 1 hash, fused. PQC counterpart of CHIPLET_PUBK_HASH1. |
| SIP_PUBK_PQC_HASH0 | 25 | 256 | true | true | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot (slot 0). PQC counterpart of SIP_PUBK_HASH0. |
| SYS_PUBK_PQC_HASH | 26 | 256 | true | true | Hash digest of the system owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SYS_PUBK_HASH. |
| SIP_PUBK_HASH1 | 27 | 256 | true | true | Hash digest of the SiP owner's root public key for secure boot, slot 1. Second slot so the SiP owner retains a usable root key after revoking slot 0 via CHIPLET_PUBK_REVOKE[24]. |
| SIP_PUBK_PQC_HASH1 | 28 | 256 | true | true | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot, slot 1. PQC counterpart of SIP_PUBK_HASH1; revocable via CHIPLET_PUBK_REVOKE[25]. |
| SEP_CHIPLET_ID | 29 | 256 | true | true | Per-owner chiplet public identity fuse used as a public identity input to the Key Manager. No read protection is required. |
| SEP_SIP_ID | 30 | 256 | true | true | SiP-owner public identity fuse used as a public identity input to the Key Manager. No read protection is required. |
| SEP_SYS_ID | 31 | 256 | true | true | System-owner public identity fuse used as a public identity input to the Key Manager. No read protection is required. |
| spare0 | 32 | 256 | true | true | Spare field 0 for future use. |
| spare1 | 33 | 256 | true | true | Spare field 1 for future use. |
| spare2 | 34 | 256 | true | true | Spare field 2 for future use. |
| spare3 | 35 | 256 | true | true | Spare field 3 for future use. |
| spare4 | 36 | 256 | true | true | Spare field 4 for future use. |
| spare5 | 37 | 256 | true | true | Spare field 5 for future use. |
| spare6 | 38 | 256 | true | true | Spare field 6 for future use. |
| spare7 | 39 | 256 | true | true | Spare field 7 for future use. |

| Field | Lock slot | Size (bits) | SW writable | SW readable | Description |
|---|---|---|---|---|---|
| spare8 | 40 | 256 | true | true | Spare field 8 for future use. |

The fuse map is exactly 8192 bits with exclusive end address `0x400` (1024 bytes, 256 × 32-bit words).

The JTAG_ID field has been removed from the fuse map and will be implemented as a top-level port.

### Anti-Rollback Security-Version Flags

`BL1_VERSION` holds the device security-version flags for BL1. They are independent flags rather than a counter, and the production ROM uses bits `[127:0]`. `BL2_VERSION` is separate storage whose interpretation belongs to later-stage firmware. Both fields are set-only, so a burned flag stays set.

The Anti-Rollback requirements of the SEP Boot ROM appendix in the Programmer's Guide define the check a manifest must pass and how firmware commits new flags.

### RTL Integration

`sep_crypto` instantiates `sep_efuse_wrapper`. The wrapper converts the 64-bit SEP AXI interface to 32-bit AXI4-Lite through `axi_dw_converter` and `axi_to_axi_lite`, then connects to the shared `efuse_interface_controller`. A separate JTAG AXI4-Lite port provides manufacturing and lifecycle-restricted access.

The controller uses `HAS_LC_STATE=1`. Its four-bit logical lifecycle state starts at bit 96 in the shadow map and uses an eight-bit differential encoding. `sep_lifecycle_ctrl` decodes that state with `prim_diff_decode_multi`, derives feature controls from `SIP_DIS` and `SYS_DIS`, and differentially encodes its two demotion outputs. The pre-sense invalid-state behavior is described under Fuse Sensing and Boot Sequencing above.

`sep_crypto` assembles the Key Manager's `km_otp_data_t` input from:

- The encoded lifecycle shadow and the lifecycle controller's two demotion outputs; demotion is not read directly from fuses.
- The secret shadows `chiplet_uid`, `sip_uid`, `sys_uid` and `class_key`.
- The public identity shadows `sep_chiplet_id`, `sep_sip_id` and `sep_sys_id`.

`BL1_VERSION`, `BL2_VERSION`, RMA token digests and public-key hashes are not part of that Key Manager interface. SEP firmware accesses them through the shadow interface; the token matcher also consumes the RMA digests internally.

For simulation, adopters need to provide an external eFuse macro model behind the shim interface. Only the generic controller and the shim interfaces are defined here.

For the canonical eFuse IP documentation covering the `efuse_interface_controller` internals, sensing FSM, and programming protocol, see `hw/ip/efuse/doc/index.adoc` (included in the SMC documentation via `hw/sys/smc/doc/periphs.adoc`).

### Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

#### SEP eFuse Map Address Map

Per-field write-lock and read-lock registers for the SEP eFuse map.

*Table 127. SEP eFuse Map Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | LOCKS | RW | - |
| 0x8 | LOCKS_SPARE | RW | - |
| 0xC | LC_STATE | R | - |
| 0x10 | SBOOT_DIS | R | - |
| 0x14 | TRANSIENT_RMA_EN | R | - |
| 0x18 | SIP_DIS | RW | An owner's feature control vector. When set (1'b1), the corresponding feature is disabled. Layout: see the Disable Vector Format section of lifecycle_controller.adoc. |
| 0x20 | SYS_DIS | RW | An owner's feature control vector. When set (1'b1), the corresponding feature is disabled. Layout: see the Disable Vector Format section of lifecycle_controller.adoc. |
| 0x28 | RMA_SIP_TOKEN_DIGEST | R | - |
| 0x48 | RMA_CHIPLET_TOKEN_DIGEST | R | - |
| 0x68 | CLASS_KEY | R | - |
| 0x88 | CHIPLET_PUBK_REVOKE | R | - |
| 0x8C | BL1_VERSION | R | - |
| 0xAC | BL2_VERSION | R | - |
| 0xCC | CHIPLET_UID | R | - |
| 0xEC | SIP_PUBK_HASH0 | R | - |
| 0x10C | SIP_UID | R | - |
| 0x12C | SYS_PUBK_HASH | R | - |
| 0x14C | SYS_UID | R | - |
| 0x16C | STATUS_RPT | R | - |
| 0x170 | ROM_CTL | R | - |
| 0x174 | SYSCLK_FREQ_MHZ | R | - |
| 0x178 | CHIPLET_PUBK_HASH0 | R | - |
| 0x198 | CHIPLET_PUBK_HASH1 | R | - |
| 0x1B8 | REQUIRED_SIGNERS | RW | - |
| 0x1BC | REQUIRED_ALGS | RW | - |
| 0x1C0 | CHIPLET_PUBK_PQC_HASH0 | R | - |
| 0x1E0 | CHIPLET_PUBK_PQC_HASH1 | R | - |
| 0x200 | SIP_PUBK_PQC_HASH0 | R | - |
| 0x220 | SYS_PUBK_PQC_HASH | R | - |
| 0x240 | SIP_PUBK_HASH1 | R | - |
| 0x260 | SIP_PUBK_PQC_HASH1 | R | - |
| 0x280 | SEP_CHIPLET_ID | R | - |
| 0x2A0 | SEP_SIP_ID | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2C0 | SEP_SYS_ID | R | - |
| 0x2E0 | spare0 | R | - |
| 0x300 | spare1 | R | - |
| 0x320 | spare2 | R | - |
| 0x340 | spare3 | R | - |
| 0x360 | spare4 | R | - |
| 0x380 | spare5 | R | - |
| 0x3A0 | spare6 | R | - |
| 0x3C0 | spare7 | R | - |
| 0x3E0 | spare8 | R | - |

### LOCKS

*Table 128. sep_efuse_map.LOCKS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | SEP_SYS_ID_READ_LOCK | RW1S | 0x0 | sep_sys_id_read_lock (slot 31): when set, SEP_SYS_ID is not read-able. |
| 62 | SEP_SYS_ID_WRITE_LOCK | RW1S | 0x0 | sep_sys_id_write_lock (slot 31): when set, SEP_SYS_ID is not write-able. |
| 61 | SEP_SIP_ID_READ_LOCK | RW1S | 0x0 | sep_sip_id_read_lock (slot 30): when set, SEP_SIP_ID is not read-able. |
| 60 | SEP_SIP_ID_WRITE_LOCK | RW1S | 0x0 | sep_sip_id_write_lock (slot 30): when set, SEP_SIP_ID is not write-able. |
| 59 | SEP_CHIPLET_ID_READ_LOCK | RW1S | 0x0 | sep_chiplet_id_read_lock (slot 29): when set, SEP_CHIPLET_ID is not read-able. |
| 58 | SEP_CHIPLET_ID_WRITE_LOCK | RW1S | 0x0 | sep_chiplet_id_write_lock (slot 29): when set, SEP_CHIPLET_ID is not write-able. |
| 57 | SIP_PUBK_PQC_HASH1_READ_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 56 | SIP_PUBK_PQC_HASH1_WRITE_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 55 | SIP_PUBK_HASH1_READ_LOCK | RW1S | 0x0 | sip_pubk_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 54 | SIP_PUBK_HASH1_WRITE_LOCK | RW1S | 0x0 | sip_pubk_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 53 | SYS_PUBK_PQC_HASH_READ_LOCK | RW1S | 0x0 | sys_pubk_pqc_hash_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 52 | SYS_PUBK_PQC_HASH_WRITE_LOCK | RW1S | 0x0 | sys_pubk_pqc_hash_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 51 | SIP_PUBK_PQC_HASH0_READ_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 50 | SIP_PUBK_PQC_HASH0_WRITE_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 49 | CHIPLET_PUBK_PQC_HASH1_READ_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 48 | CHIPLET_PUBK_PQC_HASH1_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 47 | CHIPLET_PUBK_PQC_HASH0_READ_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 46 | CHIPLET_PUBK_PQC_HASH0_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 45 | REQUIRED_ALGS_READ_LOCK | RW1S | 0x0 | required_algs_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 44 | REQUIRED_ALGS_WRITE_LOCK | RW1S | 0x0 | required_algs_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 43 | REQUIRED_SIGNERS_READ_LOCK | RW1S | 0x0 | required_signers_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 42 | REQUIRED_SIGNERS_WRITE_LOCK | RW1S | 0x0 | required_signers_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 41 | CHIPLET_PUBK_HASH1_READ_LOCK | RW1S | 0x0 | chiplet_pubk_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 40 | CHIPLET_PUBK_HASH1_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 39 | CHIPLET_PUBK_HASH0_READ_LOCK | RW1S | 0x0 | chiplet_pubk_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 38 | CHIPLET_PUBK_HASH0_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 37 | SYSCLK_FREQ_MHZ_READ_LOCK | RW1S | 0x0 | sysclk_freq_mhz_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 36 | SYSCLK_FREQ_MHZ_WRITE_LOCK | RW1S | 0x0 | sysclk_freq_mhz_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 35 | ROM_CTL_READ_LOCK | RW1S | 0x0 | rom_ctl_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 34 | ROM_CTL_WRITE_LOCK | RW1S | 0x0 | rom_ctl_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 33 | STATUS_RPT_READ_LOCK | RW1S | 0x0 | status_rpt_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 32 | STATUS_RPT_WRITE_LOCK | RW1S | 0x0 | status_rpt_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 31 | SYS_UID_READ_LOCK | RW1S | 0x0 | sys_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 30 | SYS_UID_WRITE_LOCK | RW1S | 0x0 | sys_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 29 | SYS_PUBK_HASH_READ_LOCK | RW1S | 0x0 | sys_pubk_hash_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 28 | SYS_PUBK_HASH_WRITE_LOCK | RW1S | 0x0 | sys_pubk_hash_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 27 | SIP_UID_READ_LOCK | RW1S | 0x0 | sip_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 26 | SIP_UID_WRITE_LOCK | RW1S | 0x0 | sip_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 25 | SIP_PUBK_HASH0_READ_LOCK | RW1S | 0x0 | sip_pubk_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 24 | SIP_PUBK_HASH0_WRITE_LOCK | RW1S | 0x0 | sip_pubk_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 23 | CHIPLET_UID_READ_LOCK | RW1S | 0x0 | chiplet_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 22 | CHIPLET_UID_WRITE_LOCK | RW1S | 0x0 | chiplet_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 21 | BL2_VERSION_READ_LOCK | RW1S | 0x0 | bl2_version_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 20 | BL2_VERSION_WRITE_LOCK | RW1S | 0x0 | bl2_version_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 19 | BL1_VERSION_READ_LOCK | RW1S | 0x0 | bl1_version_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 18 | BL1_VERSION_WRITE_LOCK | RW1S | 0x0 | bl1_version_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 17 | CHIPLET_PUBK_REVOKE_READ_LOCK | RW1S | 0x0 | chiplet_pubk_revoke_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 16 | CHIPLET_PUBK_REVOKE_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_revoke_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 15 | CLASS_KEY_READ_LOCK | RW1S | 0x0 | class_key_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 14 | CLASS_KEY_WRITE_LOCK | RW1S | 0x0 | class_key_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 13 | RMA_CHIPLET_TOKEN_DIGEST_READ_LOCK | RW1S | 0x0 | rma_chiplet_token_digest_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 12 | RMA_CHIPLET_TOKEN_DIGEST_WRITE_LOCK | RW1S | 0x0 | rma_chiplet_token_digest_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 11 | RMA_SIP_TOKEN_DIGEST_READ_LOCK | RW1S | 0x0 | rma_sip_token_digest_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 10 | RMA_SIP_TOKEN_DIGEST_WRITE_LOCK | RW1S | 0x0 | rma_sip_token_digest_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 9 | SYS_DIS_READ_LOCK | RW1S | 0x0 | sys_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 8 | SYS_DIS_WRITE_LOCK | RW1S | 0x0 | sys_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 7 | SIP_DIS_READ_LOCK | RW1S | 0x0 | sip_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 6 | SIP_DIS_WRITE_LOCK | RW1S | 0x0 | sip_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 5 | TRANSIENT_RMA_EN_READ_LOCK | RW1S | 0x0 | TRANSIENT_RMA_EN_READ_LOCK, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 4 | TRANSIENT_RMA_EN_WRITE_LOCK | RW1S | 0x0 | TRANSIENT_RMA_EN_WRITE_LOCK, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 3 | SBOOT_DIS_READ_LOCK | RW1S | 0x0 | sboot_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 2 | SBOOT_DIS_WRITE_LOCK | RW1S | 0x0 | sboot_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 1 | LC_STATE_READ_LOCK | RW1S | 0x0 | lc_state_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 0 | LC_STATE_WRITE_LOCK | RW1S | 0x0 | lc_state_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |

### LOCKS_SPARE

*Table 129. sep_efuse_map.LOCKS_SPARE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:18 | SPARE_LOCK_RSVD | RW1S | 0x0 | Unassigned lock slots 41-47. Reserved for future field lock assignments. |
| 17 | SPARE8_READ_LOCK | RW1S | 0x0 | spare8_read_lock (slot 40): when set, spare8 is not read-able. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 16 | SPARE8_WRITE_LOCK | RW1S | 0x0 | spare8_write_lock (slot 40): when set, spare8 is not write-able. |
| 15 | SPARE7_READ_LOCK | RW1S | 0x0 | spare7_read_lock (slot 39): when set, spare7 is not read-able. |
| 14 | SPARE7_WRITE_LOCK | RW1S | 0x0 | spare7_write_lock (slot 39): when set, spare7 is not write-able. |
| 13 | SPARE6_READ_LOCK | RW1S | 0x0 | spare6_read_lock (slot 38): when set, spare6 is not read-able. |
| 12 | SPARE6_WRITE_LOCK | RW1S | 0x0 | spare6_write_lock (slot 38): when set, spare6 is not write-able. |
| 11 | SPARE5_READ_LOCK | RW1S | 0x0 | spare5_read_lock (slot 37): when set, spare5 is not read-able. |
| 10 | SPARE5_WRITE_LOCK | RW1S | 0x0 | spare5_write_lock (slot 37): when set, spare5 is not write-able. |
| 9 | SPARE4_READ_LOCK | RW1S | 0x0 | spare4_read_lock (slot 36): when set, spare4 is not read-able. |
| 8 | SPARE4_WRITE_LOCK | RW1S | 0x0 | spare4_write_lock (slot 36): when set, spare4 is not write-able. |
| 7 | SPARE3_READ_LOCK | RW1S | 0x0 | spare3_read_lock (slot 35): when set, spare3 is not read-able. |
| 6 | SPARE3_WRITE_LOCK | RW1S | 0x0 | spare3_write_lock (slot 35): when set, spare3 is not write-able. |
| 5 | SPARE2_READ_LOCK | RW1S | 0x0 | spare2_read_lock (slot 34): when set, spare2 is not read-able. |
| 4 | SPARE2_WRITE_LOCK | RW1S | 0x0 | spare2_write_lock (slot 34): when set, spare2 is not write-able. |
| 3 | SPARE1_READ_LOCK | RW1S | 0x0 | spare1_read_lock (slot 33): when set, spare1 is not read-able. |
| 2 | SPARE1_WRITE_LOCK | RW1S | 0x0 | spare1_write_lock (slot 33): when set, spare1 is not write-able. |
| 1 | SPARE0_READ_LOCK | RW1S | 0x0 | spare0_read_lock (slot 32): when set, spare0 is not read-able. |
| 0 | SPARE0_WRITE_LOCK | RW1S | 0x0 | spare0_write_lock (slot 32): when set, spare0 is not write-able. |

### LC_STATE

*Table 130. sep_efuse_map.LC_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | rsvd | R | 0x0 | Unused |
| 7:0 | lc_state | R | 0xF0 | Life cycle manufacturing state, differentially encoded: {~raw[3:0], raw[3:0]} |

### SBOOT_DIS

*Table 131. sep_efuse_map.SBOOT_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | rsvd | R | 0x0 | Unused |
| 0 | disable_secure_boot | R | 0x0 | Chicken bit to disable secure boot in SEP ROM. |

### TRANSIENT_RMA_EN

*Table 132. sep_efuse_map.TRANSIENT_RMA_EN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | rsvd | R | 0x0 | Unused |
| 0 | transient_rma_en | R | 0x0 | When Transient RMA EN is 1, matching on corresponding *_Tokens will automatically transfer the chip to the corresponding RMA state by setting the LC_STATE reg |

### SIP_DIS

*Table 133. sep_efuse_map.SIP_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:48 | `func_reserved` | RW1S | 0x0 | Reserved for adopter-defined functional feature gating (bits 63:48). |
| 47:25 | `debug_reserved_dbg2` | RW1S | 0x0 | Reserved for adopter-defined inter-chiplet-bypass (DBG_2) debug scan chains (bits 47:25). |
| 24 | `sip_debug` | RW1S | 0x0 | SIP_DBG (bit 24): Coarse-grained debug enable for the SiP scope: the inter-chiplet debug-bypass gate that admits external (PTAP) debug traffic into this chiplet and forwards it onward to the next chiplet via the outbound JTAG PTAP. 1'b1: disabled. 1'b0: enabled. |
| 23:4 | `debug_reserved_dbg1` | RW1S | 0x0 | Reserved for adopter-defined local (DBG_1) debug scan chains (bits 23:4). |
| 3 | `smc_fuse_dbg` | RW1S | 0x0 | SMC_FUSE_DBG (bit 3): 1'b1: the DFT-inserted access path into the SMC OTP is disabled. 1'b0: enabled. Produced by SEP-LCC feat_ctrl_o, exported on the SEP boundary, and consumed by SMC to gate its own inserted fuse path. |
| 2 | `sep_fuse_dbg` | RW1S | 0x0 | SEP_FUSE_DBG (bit 2): 1'b1: the DFT-inserted access path into the SEP OTP is disabled. 1'b0: enabled. Also covers the OTP IP-vendor test and calibration interface, which is reachable only over that inserted path. The functional JTAG2AXIL path to the OTP is always available and is not gated by this bit. |
| 1 | `chiplet_dbg` | RW1S | 0x0 | CHIPLET_DBG (bit 1): Coarse-grained debug enable for the chiplet scope. 1'b1: chiplet-scope debug/scan-dump access disabled. 1'b0: enabled. Never acts alone: an external debugger reaches chiplet-internal resources only when SIP_DBG is open as well. Contributes to the reachability of the crypto/KM secure chain but never authorizes a dump from it. |
| 0 | `sep_debug` | RW1S | 0x0 | SEP_DBG (bit 0): Coarse-grained debug enable for the SEP scope. 1'b1: SEP-scope debug/scan-dump access disabled and external inbound traffic is filtered. 1'b0: enabled. A local gate only: reaching SEP internal state from an external debugger additionally requires SIP_DBG and CHIPLET_DBG. |

### SYS_DIS

*Table 134. sep_efuse_map.SYS_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:48 | `func_reserved` | RW1S | 0x0 | Reserved for adopter-defined functional feature gating (bits 63:48). |
| 47:25 | `debug_reserved_dbg2` | RW1S | 0x0 | Reserved for adopter-defined inter-chiplet-bypass (DBG_2) debug scan chains (bits 47:25). |
| 24 | `sip_debug` | RW1S | 0x0 | SIP_DBG (bit 24): Coarse-grained debug enable for the SiP scope: the inter-chiplet debug-bypass gate that admits external (PTAP) debug traffic into this chiplet and forwards it onward to the next chiplet via the outbound JTAG PTAP. 1'b1: disabled. 1'b0: enabled. |
| 23:4 | `debug_reserved_dbg1` | RW1S | 0x0 | Reserved for adopter-defined local (DBG_1) debug scan chains (bits 23:4). |
| 3 | `smc_fuse_dbg` | RW1S | 0x0 | SMC_FUSE_DBG (bit 3): 1'b1: the DFT-inserted access path into the SMC OTP is disabled. 1'b0: enabled. Produced by SEP-LCC feat_ctrl_o, exported on the SEP boundary, and consumed by SMC to gate its own inserted fuse path. |
| 2 | `sep_fuse_dbg` | RW1S | 0x0 | SEP_FUSE_DBG (bit 2): 1'b1: the DFT-inserted access path into the SEP OTP is disabled. 1'b0: enabled. Also covers the OTP IP-vendor test and calibration interface, which is reachable only over that inserted path. The functional JTAG2AXIL path to the OTP is always available and is not gated by this bit. |
| 1 | `chiplet_dbg` | RW1S | 0x0 | CHIPLET_DBG (bit 1): Coarse-grained debug enable for the chiplet scope. 1'b1: chiplet-scope debug/scan-dump access disabled. 1'b0: enabled. Never acts alone: an external debugger reaches chiplet-internal resources only when SIP_DBG is open as well. Contributes to the reachability of the crypto/KM secure chain but never authorizes a dump from it. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `sep_debug` | RW1S | 0x0 | SEP_DBG (bit 0): Coarse-grained debug enable for the SEP scope. 1'b1: SEP-scope debug/scan-dump access disabled and external inbound traffic is filtered. 1'b0: enabled. A local gate only: reaching SEP internal state from an external debugger additionally requires SIP_DBG and CHIPLET_DBG. |

### RMA_SIP_TOKEN_DIGEST

*Table 135. sep_efuse_map.RMA_SIP_TOKEN_DIGEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `token_digest` | R | 0x0 | Hash digest of life cycle secret that allows entering RMA_SIP state / Note this is a 256 bit field |

### RMA_CHIPLET_TOKEN_DIGEST

*Table 136. sep_efuse_map.RMA_CHIPLET_TOKEN_DIGEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `token_digest` | R | 0x0 | Hash digest of life cycle secret that allows entering RMA_CHIPLET state/ Note this is a 256 bit field |

### CLASS_KEY

*Table 137. sep_efuse_map.CLASS_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key` | R | 0x0 | Class key that could be used for FW encryption/decryption |

### CHIPLET_PUBK_REVOKE

*Table 138. sep_efuse_map.CHIPLET_PUBK_REVOKE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `select` | R | 0x0 | Global public-key revocation vector covering every boot-verification key of every owner (classical and PQC). SEP ROM embeds a set of public-key hash digests and accesses additional fused key hashes; each key entry maps to one bit here. Once set, SEP ROM rejects the corresponding key. Bit map: [7:0] ROM chiplet-creator classical keys (bits 0,1 are development keys); [15:8] ROM chiplet-creator PQC keys; [16]/[17] CHIPLET_PUBK_HASH0/1; [18]/[19] CHIPLET_PUBK_PQC_HASH0/1; [20]/[21] SIP_PUBK_HASH0 / SIP_PUBK_PQC_HASH0; [22]/[23] SYS_PUBK_HASH / SYS_PUBK_PQC_HASH; [24]/[25] SIP_PUBK_HASH1 / SIP_PUBK_PQC_HASH1; [31:26] reserved. Before the chiplet owner transfers a part to the customer, the development-key bits shall be set (revoked). |

### BL1_VERSION

*Table 139. sep_efuse_map.BL1_VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `version` | R | 0x0 | Minimum BL1 version that can be loaded by SEPROM |

### BL2_VERSION

*Table 140. sep_efuse_map.BL2_VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `version` | R | 0x0 | Minimum BL2 version that can be loaded by BL1. If more boot stages exist, rollback protection can be implemented by BL2. |

### CHIPLET_UID

*Table 141. sep_efuse_map.CHIPLET_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | Silicon creator's secret device unique ID (also referred to as UDS - Unique Device Secret). The chiplet creator is responsible for CHIPLET_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### SIP_PUBK_HASH0

*Table 142. sep_efuse_map.SIP_PUBK_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of SiP owner's root public key for secure boot. The chiplet creator's actual root public key is embedded in the SEPROM; this digest is used to validate that public key. |

### SIP_UID

*Table 143. sep_efuse_map.SIP_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | SiP owner's secret device unique ID. The SiP owner is responsible for SIP_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### SYS_PUBK_HASH

*Table 144. sep_efuse_map.SYS_PUBK_HASH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of system owner's root public key for secure boot. |

### SYS_UID

*Table 145. sep_efuse_map.SYS_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | System owner's secret device unique ID. The system owner is responsible for SYS_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### STATUS_RPT

*Table 146. sep_efuse_map.STATUS_RPT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `reserved` | R | 0x0 | reserved |
| 1:0 | `rpt` | R | 0x0 | BOOTROM report status based on the bit. 2'b00→ bootrom report status based on GPIO; 2'b01→bootrom report status;otherwise, bootrom does not report status |

### ROM_CTL

*Table 147. sep_efuse_map.ROM_CTL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `reserved` | R | 0x0 | reserved |
| 5:1 | `rom_swap_ctrl` | R | 0x0 | controls how the rom slices are swapped |
| 0 | `rom_endianness_ctrl` | R | 0x0 | controls the endianness of the rom data out. 0→little endian; 1→big endian |

### SYSCLK_FREQ_MHZ

*Table 148. sep_efuse_map.SYSCLK_FREQ_MHZ fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `rsvd` | R | 0x0 | Reserved |
| 10:0 | `sysclk_freq_mhz` | R | 0x0 | Configured sysclk PLL frequency in MHz; ignored if 0. Consumed by boot ROM clock math. |

### CHIPLET_PUBK_HASH0

*Table 149. sep_efuse_map.CHIPLET_PUBK_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner authentication key hash digest, fused. These chiplet owner fused key hashes can be used in addition to the hashes in the ROM itself to authenticate manifests. |

### CHIPLET_PUBK_HASH1

*Table 150. sep_efuse_map.CHIPLET_PUBK_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner authentication key hash digest, fused. These chiplet owner fused key hashes can be used in addition to the hashes in the ROM itself to authenticate manifests. |

### REQUIRED_SIGNERS

*Table 151. sep_efuse_map.REQUIRED_SIGNERS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `reserved` | R | 0x0 | Reserved |
| 1:0 | `required_signers` | RW1S | 0x0 | Owners whose signature is required to authenticate any image loadable by SEP ROM. Per owner, 1 = required, 0 = not-required; the default all-zero value requires no additional owner. [0] SiP owner, [1] system owner. The chiplet creator is the root of trust and always authenticates BL1, so it has no bit here. |

### REQUIRED_ALGS

*Table 152. sep_efuse_map.REQUIRED_ALGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `reserved` | R | 0x0 | Reserved |
| 11:8 | `sys_algs` | RW1S | 0x0 | Per-owner required signature algorithms for system owner. bit 0 = classical, bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |
| 7:4 | `sip_algs` | RW1S | 0x0 | Per-owner required signature algorithms for SiP owner. bit 0 = classical, bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |
| 3:0 | `chiplet_algs` | RW1S | 0x0 | Per-owner required signature algorithms for chiplet. Each nibble: bit 0 = classical (ECDSA P-384 or RSA), bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |

### CHIPLET_PUBK_PQC_HASH0

*Table 153. sep_efuse_map.CHIPLET_PUBK_PQC_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner ML-DSA PQC authentication key hash digest, fused. PQC counterpart of the corresponding CHIPLET_PUBK_HASH. |

### CHIPLET_PUBK_PQC_HASH1

*Table 154. sep_efuse_map.CHIPLET_PUBK_PQC_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner ML-DSA PQC authentication key hash digest, fused. PQC counterpart of the corresponding CHIPLET_PUBK_HASH. |

### SIP_PUBK_PQC_HASH0

*Table 155. sep_efuse_map.SIP_PUBK_PQC_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SIP_PUBK_HASH. |

### SYS_PUBK_PQC_HASH

*Table 156. sep_efuse_map.SYS_PUBK_PQC_HASH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of the system owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SYS_PUBK_HASH. |

### SIP_PUBK_HASH1

*Table 157. sep_efuse_map.SIP_PUBK_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of SiP owner's root public key for secure boot. The chiplet creator's actual root public key is embedded in the SEPROM; this digest is used to validate that public key. |

### SIP_PUBK_PQC_HASH1

*Table 158. sep_efuse_map.SIP_PUBK_PQC_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SIP_PUBK_HASH. |

### SEP_CHIPLET_ID

*Table 159. sep_efuse_map.SEP_CHIPLET_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `id` | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### SEP_SIP_ID

*Table 160. sep_efuse_map.SEP_SIP_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `id` | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### SEP_SYS_ID

*Table 161. sep_efuse_map.SEP_SYS_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `id` | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### spare0

*Table 162. sep_efuse_map.spare0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare1

*Table 163. sep_efuse_map.spare1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare2

*Table 164. sep_efuse_map.spare2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare3

*Table 165. sep_efuse_map.spare3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare4

*Table 166. sep_efuse_map.spare4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare5

*Table 167. sep_efuse_map.spare5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare6

*Table 168. sep_efuse_map.spare6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare7

*Table 169. sep_efuse_map.spare7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare8

*Table 170. sep_efuse_map.spare8 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

#### Token Processing

SEP uses 256-bit tokens to authorize RMA lifecycle transitions and emergency security disable. The OTP controller hashes each submitted token with SHA-256 and compares the digest with the expected value. Plaintext tokens are held in three separate, non-scannable eight-word arrays in the eFuse MMR block. These arrays have no reset; reset is not a token-erasure operation.

#### Token Generation, Storage and Provisioning

Generate and retain plaintext tokens within a hardware security module (HSM), using its secure random-generation and derivation facilities. Keep token processing inside the HSM except for final provisioning, and protect transfer to the manufacturing tester. Provision the RMA token digests, not plaintext tokens, into OTP at manufacturing test. Provisioning policy may use a common token or a distinct token per device; the matching mechanism supports either.

Track each token and its SHA-256 digest separately. Token confidentiality and entropy protect authorization; expected-digest integrity prevents substitution of a digest for a token an attacker knows. The hardware-embedded SEC_DIS digest is common to devices using that design. Bind it under controlled implementation procedures; it does not require the plaintext token in the RTL database.

#### Token Matching Mechanism and Attack Countermeasures

*Table 171. Token roles and expected digests*

| Token | Purpose | Expected digest |
|---|---|---|
| RMA_SIP_TOKEN | Authorize SiP RMA transitions | `RMA_SIP_TOKEN_DIGEST` in OTP |
| RMA_CHIPLET_TOKEN | Authorize chiplet RMA transitions | `RMA_CHIPLET_TOKEN_DIGEST` in OTP |
| SEC_DIS_TOKEN | Emergency debug access to a part whose fuse sensing fails | Hardware constant, independent of OTP |

Each token has a dedicated SHA-256 engine and three redundant digest comparators:

```
match = (SHA256(token[255:0]) == expected_digest[255:0])
```

Hashing is sequential; the digest comparison is combinational. The computed digest and its valid bit survive reset. A new start clears validity; hash completion replaces the digest. Test enable freezes those retained latches. The presented token and expected digest must not enter scannable storage.

RMA comparisons require both a valid computed digest and completed fuse sensing, because their expected values come from OTP shadows. SEC_DIS requires only a valid digest: its expected value is implemented in gates, so a SEC_DIS match holds before fuse sensing completes. A match forces feature control on, stops fuse sensing, and opens eFuse shadow-map reads. It does not release SEP reset; the IC_RESET `sep_reset_n` override does. The SEC_DIS recovery sequence is in the Programmer's Guide. SEC_DIS follows the feature-control rules in the Lifecycle Controller chapter, with the metal-programmable enable also required. The design requirement is A0-only use and permanent removal by a metal-mask ECO in later A1/B0 revisions; this is not a software-disable mechanism. See Security Disable (SEC_DIS).

#### Token Submission

SEP software or JTAG submits tokens through the eFuse MMR interface, subject to its access policy. Register names and start bits are:

*Table 172. Token submission registers*

| Eight-word input array | TOKEN_EOP bit | Result register |
|---|---|---|
| RMA_SIP_TOKEN_I | 0 | RMA_SIP_TOKEN_MATCH |
| RMA_CHIPLET_TOKEN_I | 8 | RMA_CHIPLET_TOKEN_MATCH |
| SEC_DISABLE_TOKEN_I | 16 | SEC_DISABLE_TOKEN_MATCH |

Index 0 of an input array holds `token[31:0]` and index 7 holds `token[255:224]`; the hash engine consumes index 7 first, down to index 0, without a byte swap. A write of 1 to a `TOKEN_EOP` bit generates a start pulse, and the engine then streams the array rather than taking a snapshot. RMA results remain idle until fuse sensing completes. The token submission sequence is in the Programmer's Guide.

There is no submission queue: the feeder accepts a new operation only while idle. Starting a hash clears retained digest validity, so comparison returns idle while the new digest is unavailable. The result register follows the comparator through a clocked output stage; writing the token words alone does not invalidate the previous result or launch a comparison.

*Table 173. Token result encodings*

| Result | Six-bit value | Meaning |
|---|---|---|
| Idle | `0x00` (`000000`) | No valid comparison; hashing or fuse sensing may be pending |
| Match | `0x15` (`010101`) | All three comparators report equal digests |
| Mismatch | `0x2a` (`101010`) | All three comparators report unequal digests |
| Error | `0x3f` (`111111`) | A differential-pair or comparator-agreement fault is present |

Reset clears the registered result and sticky fault flags, but does not erase the input arrays, retained digest or digest-valid state. A retained digest can therefore produce a result again after reset once its comparison prerequisites hold, so reset alone does not revoke a prior token authorization.

#### Matching Logic Countermeasures

The three comparators provide differential outputs. When comparison is valid, each pair must contain opposite values and all three instances must agree. A collapsed pair or disagreement raises a redundancy fault and forces the current comparator result to `0x3f`, which does not satisfy a match check. This is not majority voting: two matching instances cannot overrule a third.

Each fault also sets a software-read-only flag in `TOKEN_MATCH_FAULT`: bit 0 for SiP RMA, bit 8 for chiplet RMA and bit 16 for SEC_DIS. These flags remain set until reset and combine into SEP CPU PIC source 40; there is no write-one-to-clear operation to deassert the source. See SEP CPU Interrupt Vector Map.

The sticky flag records fault history; it does not latch the comparator into an error state. A later valid comparison can report match or mismatch while the fault flag and interrupt remain asserted.

The detector does not cover common-mode faults. If all three instances flip in the same way while retaining legal differential pairs, they still agree. Inverting a mismatch can therefore produce an undetected match (fail-open); inverting a match produces a mismatch. The countermeasure detects collapsed pairs and disagreement, not every possible fault-injection outcome.

Figure 27 shows the processing path. The six-bit RMA match codes are checked by the lifecycle and fuse-programming consumers. SEC_DIS has a separate bypass output qualified by the revision enable.

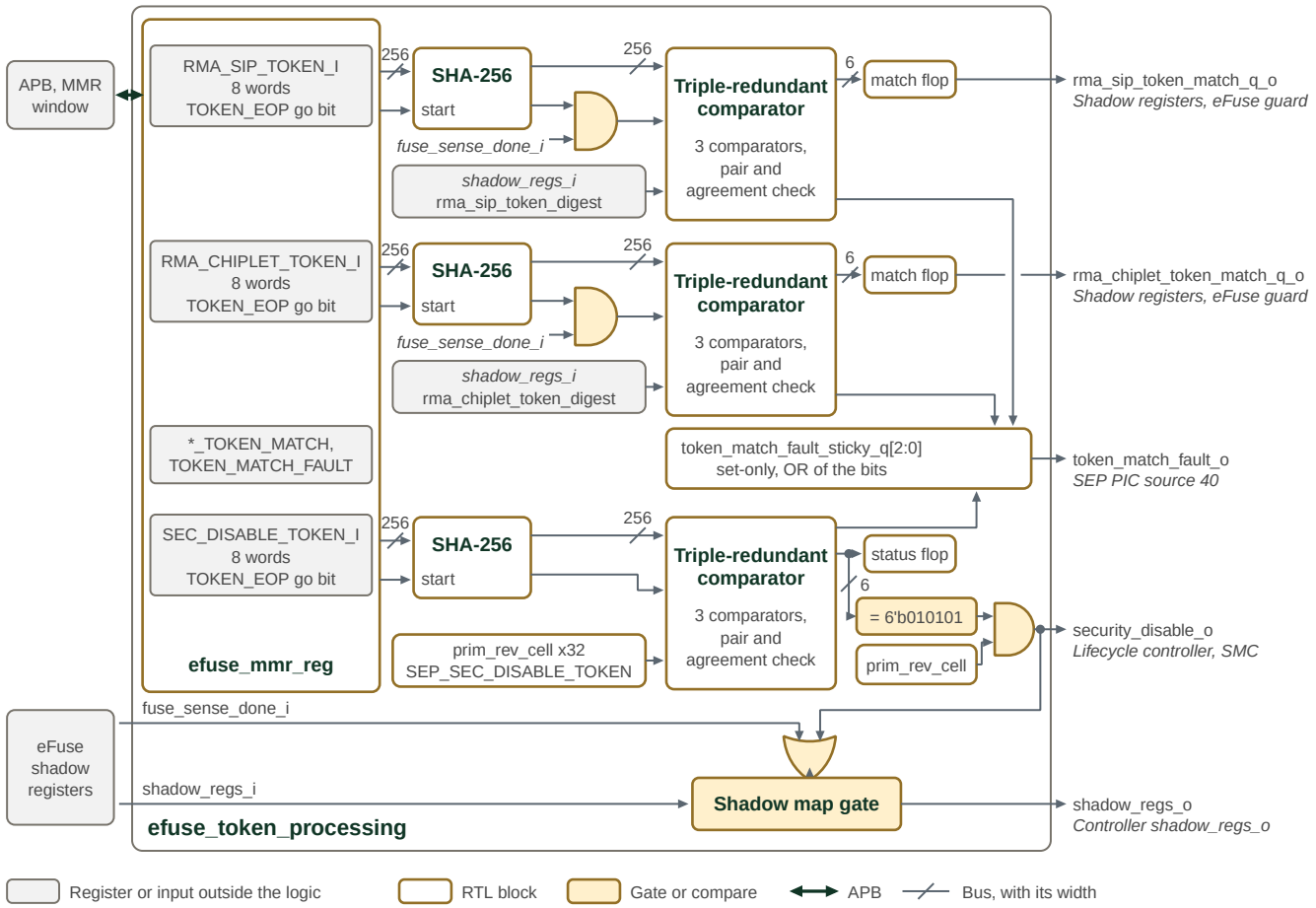


*Figure 27. SEP token processing*

Each of the three rows in Figure 27 is one token lane, from its token array and go bit in `efuse_mmr_reg`, through its SHA-256 engine, to its comparator. The expected digests and `fuse_sense_done_i` are named at their point of use; both come from the eFuse shadow registers at the lower left. `fuse_sense_done_i` gates only the two RMA comparators. The comparators' match codes and the sticky fault bits are also readable in the `*_TOKEN_MATCH` and `TOKEN_MATCH_FAULT` status registers. The RMA_CHIPLET match line crosses the RMA_SIP fault line without connecting; the gap marks the crossing.

### Post Token Matching Handling

An RMA match authorizes a lifecycle change; it does not itself burn a fuse. The controller gates RMA fuse programming with the corresponding token match. For shadow-state transitions, software can request an explicit update, or `TRANSIENT_RMA_EN` can enable automatic updates. The chiplet-RMA shadow transition also requires RMA_SIP to be established first. See OTP Fuse Controller for lock exceptions and transition rules. SEC_DIS instead activates the emergency bypass while its match and revision-enable conditions hold.

### Security Disable (SEC_DIS)

Security Disable (SEC_DIS) is an emergency bring-up bypass for early (A0) silicon. On deassertion of power-on reset, the fuse sense logic runs before anything else, and a malfunction there — the fuse state machine hanging, or invalid values being written into the shadow register bank — would leave the life-cycle control closed and turn the chiplet into a disabled ``brick''. SEC_DIS provides a secure unlock mechanism to recover such a part for debug: it overrides the life-cycle control, forcing all feature-control bits on regardless of the LC state and the SIP_DIS / SYS_DIS disables.

SECURE_TM is not a term in the feature-control computation, so SEC_DIS neither qualifies nor overrides it. In particular,

SEC_DIS does not open the secure scan chain to a dump: that chain admits only test commands, and only at `secure_tm = 1`, independently of the feature-control bits (see the Lifecycle Controller chapter, *`secure_tm` and the Secure Scan Chain*).

#### Token Activation

SEC_DIS is activated by presenting a secret token: SEC_DIS_TOKEN_I is passed into the SEP via the JTAG port, hashed, and redundantly compared against an embedded hash constant. On a match, i.e. hash(SEC_DIS_TOKEN_I) == SEP_SEC_DISABLE_TOKEN_DIGEST, the life-cycle control is overridden to open debug access, and the feature-control output follows the SEC_DIS rules in the Lifecycle Controller chapter. A match does not release the SEP reset, which stays held until fuse sensing completes. To boot a part whose fuse sensing does not complete, override `sep_reset_n` through the SEP slice of the IC_RESET TDR (see IC_RESET Overrides in the DTP chapter). The override and the token match are independent JTAG steps: the override alone releases the reset but leaves the life-cycle control closed, so recovering such a part for debug takes both.

#### Fuse Sensing and the Shadow Map

While a SEC_DIS match holds:

- Automatic fuse sensing does not start, and sensing already in progress stops. No further fuse words are loaded into the shadow registers, and fuse sensing does not complete.
- A read of the eFuse shadow map over the SEP bus completes with OKAY.
- The SMC eFuse controller receives the SEP security-disable status, so SMC fuse sensing stops and the SMC shadow map opens in the same way.

#### Reset Persistence

The token arrays, the computed digest and its valid bit survive reset, so a match established before a cold reset holds from cold-reset release. Fuse sensing then does not start, every shadow register keeps its reset value instead of the partial contents of an interrupted sense, and SEP samples the secure test-mode request one cycle after cold-reset release instead of at fuse-sensing completion. A match established without a following cold reset leaves the shadow registers holding whatever the failed sense loaded, and the secure test-mode request is never sampled. The SEC_DIS recovery sequence is in the Programmer's Guide.

#### Silicon Revision Scope

SEC_DIS is intended for A0 silicon only and is removed in later revisions (A1/B0) by a simple metal-mask ECO: the token comparator's enable is implemented in a metal-programmable revision cell, strapped to 1 (comparator enabled) in A0 silicon and re-strapped to 0 at top metal in later revisions, so the feature is killed without a re-spin.

#### Implementation Requirements

- The token secret itself is never stored in RTL; the RTL stores only the token hash value, and the hardware forces every presented token through the hash engine before the digest comparison — there is no path that presents a digest to the comparator directly. The 256-bit SEP_SEC_DISABLE_TOKEN_DIGEST constant is therefore not itself a secret (recovering the token from it requires a SHA-256 preimage): it may be publicly known and may be carried in plain RTL. It resides in the fuse controller component hardware gates (*not the OTP fuse bank*).
- The comparison logic must be purely combinational, and the SEP_SEC_DISABLE_TOKEN_DIGEST constant must never be latched into any potentially scannable register. This is an integrity requirement, not confidentiality: the expected digest must not be perturbable — for example scan-shifted — into a value whose preimage an attacker knows.

- The JTAG-accessible register holding the token input values (SEC_DIS_TOKEN_I, RMA_SIP_TOKEN_I, and RMA_CHIPLET_TOKEN_I) ***shall not have a reset***, so that the SEC_DIS state is retained across reset for early-event debug. That register must also not be scannable: DFT scan tests may be running while SEC_DIS is asserted, and a scan pattern must not inadvertently overwrite the register — dropping the security unlock and stopping the test.
- The secrecy of the scheme rests entirely on the SEC_DIS_TOKEN value, never on the hash. The token shall be a full-entropy 256-bit random value, and knowledge of it shall be strictly controlled: it shall be generated and stored in a secure database (HSM). Because only its integrity matters, the SEP_SEC_DISABLE_TOKEN_DIGEST may be bound at any convenient point in the implementation flow — for example as an RTL parameter passed down the instance hierarchy (chiplet top to SMU to SEP to fuse controller). One member of the DV team shall perform final verification of the SEC_DIS_TOKEN functionality on the post-layout extracted netlist with ***gate-level simulation***.

### Life Cycle Controller (LCC)

The Life Cycle Controller (LCC) converts the OTP lifecycle state and owner restrictions into `feat_ctrl_o`, the 64-bit vector that enables debug and functional features. Two volatile demotion controls can change the debug profile for the current boot without changing OTP.

The OTP controller supplies `LC_STATE`, `SIP_DIS` and `SYS_DIS` after fuse sensing. The LCC checks the differentially encoded state and disables features on an integrity error, except when the final `SEC_DIS` override forces them on. Downstream logic uses these outputs to gate debug, test and functional paths.

The sections below explain states and security domains, feature-control rules, demotion, scan protection and firmware authorization. Hardware gating is implemented; the multi-owner signing model in Firmware Authorization is a deferred firmware design. Do not use that model as a description of current ROM verification.

Figure 28 shows a high-level view of the Life Cycle Controller components.

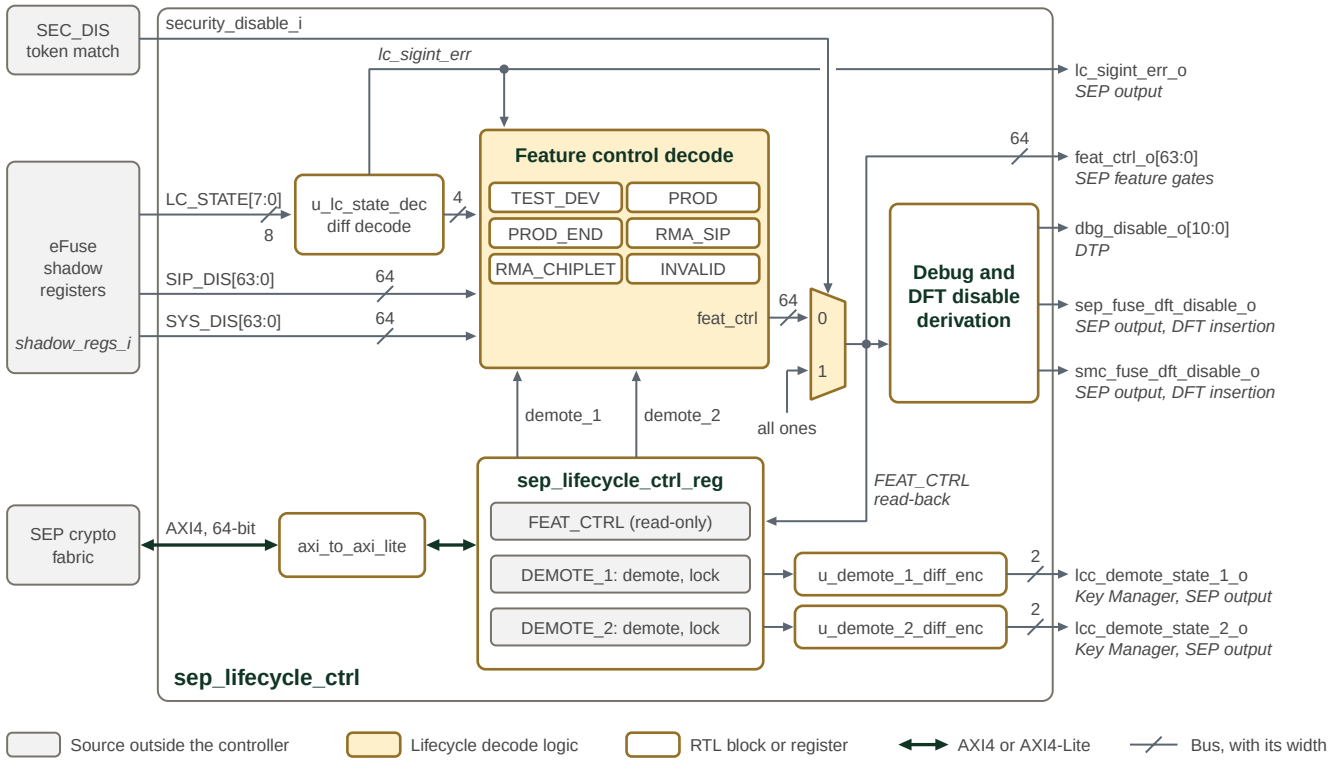


*Figure 28. SEP lifecycle controller*

In Figure 28, the inputs enter on the left and the outputs leave on the right, each labelled with its consumer.

`u_lc_state_dec` decodes the differentially encoded LC_STATE from the eFuse shadow registers and raises `lc_sigint_err` on a non-complementary codeword, which forces the feature control decode to all zeros. While `security_disable_i` is high, the multiplexer replaces the decoded vector with all ones, even on an integrity error. The `lc_sigint_err` line crosses the `security_disable_i` line without connecting; the gap marks the crossing.

The debug and DFT disable derivation turns the `sip_debug`, `chiplet_dbg`, `sep_debug`, `sep_fuse_dbg` and `smc_fuse_dbg` bits of the result into the 11 debug disables for the DTP and the two fuse DFT disables. No functional logic consumes the fuse DFT disables; they leave the SMU for adopter DFT insertion. The `demote` and `lock` bits of DEMOTE_1 and DEMOTE_2 are set-only: firmware can set `demote` only while `lock` is clear, and both hold until SEP cold reset. The demote bits feed the feature control decode and, through the differential encoders, the Key Manager and the SEP outputs.

**Lifecycle States**

The LCC decodes the OTP controller's `LC_STATE` shadow output; it does not sequence lifecycle transitions itself. Before fuse sensing completes, that output holds INVALID (`4'b1111`). The LCC then follows the sensed state and its feature-control profile.

The raw four-bit OTP value is carried as a complementary code in the shadow path. A failed integrity check sets `lc_sigint_err` and forces the base feature vector to zero. `SEC_DIS` is applied afterward and can override that result; see Overrides and Final Gates (SEC_DIS).

Hardware orders the RMA token transitions: `RMA_SIP_TOKEN` first admits RMA_SIP, then `RMA_CHIPLET_TOKEN` admits RMA_CHIPLET. Volatile demotion is separate from those transitions and does not change the OTP state.

Key Manager includes DEMOTE_1/2 with CHIPLET_UID, SIP_UID, SYS_UID and LC_STATE in key derivation, separating demoted-state keys from non-demoted production keys.

Figure 29 shows the lifecycle states and transition paths.

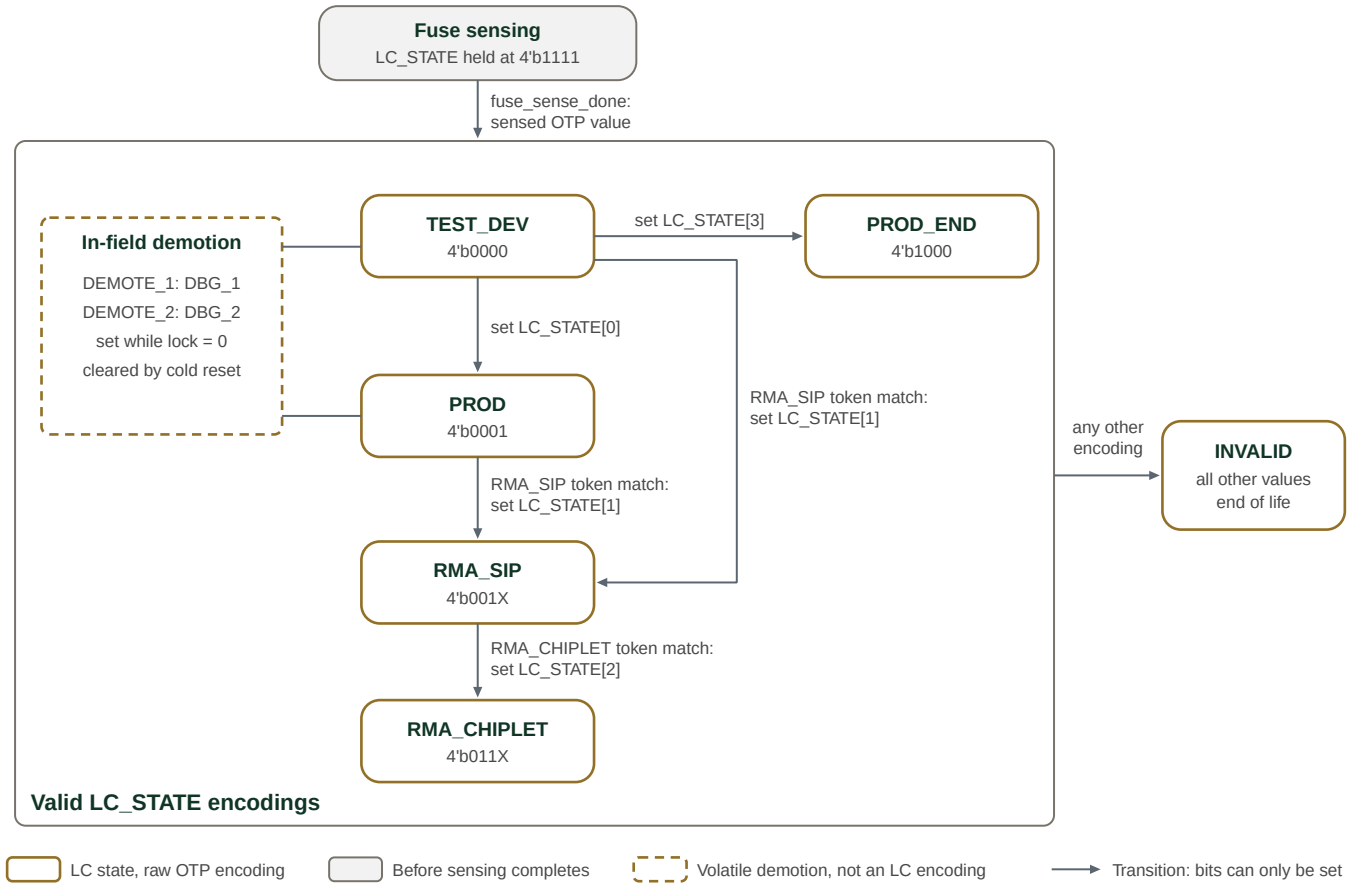


*Figure 29. SEP lifecycle state machine*

Each rounded box in Figure 29 is a lifecycle state with its raw OTP encoding, where X marks a bit that does not change the state. LC_STATE bits can only be set, so each arrow names the bit that it sets. Setting LC_STATE[1] requires an RMA_SIP token match. Setting LC_STATE[2] requires an RMA_CHIPLET token match and a state that already has LC_STATE[1] set. Every other encoding is INVALID and permanent: for example, setting LC_STATE[3] in any state other than TEST_DEV, or setting any further bit in PROD_END. Before fuse sensing completes, the shadow output holds the INVALID encoding 4'b1111. The dashed box is the volatile demotion overlay, which acts in TEST_DEV and PROD without changing the encoding.

Table 174 lists the raw OTP encodings. The complementary shadow encoding is a transport integrity check, not an additional lifecycle state.

*Table 174. SEP lifecycle state encodings*

| LC State Name | Encoding | Description |
|---|---|---|
| TEST_DEV | 4'b0000 | Default silicon state at manufacturing time.<br><br>The chip is open for ATE/SLT testing, fuse provisioning, and mission-mode software development, with certain chiplet features disabled as defined by the chiplet creator, the SiP owner, and the system owner.<br><br>In this state, the hash digests of the secret tokens (RMA_CHIPLET_TOKEN and RMA_SIP_TOKEN) are programmed into OTP by the chiplet creator and the SiP owner, respectively.<br><br>The disable vectors (SIP_DIS and SYS_DIS) are likewise programmed into OTP by the chiplet creator and the SiP owner. |

| LC State Name | Encoding | Description |
|---|---|---|
| PROD | 4'b0001 | Product functioning state. In-field debug is enabled by the volatile DEMOTE_1 / DEMOTE_2 registers (not a separate LC encoding): DEMOTE_1 opens the local debug group (DBG_1) and DEMOTE_2 opens the inter-chiplet debug-bypass group (DBG_2).<br><br>See In-Field Debug Demotion. |
| PROD_END | 4'b1000 | Terminal product state; the only permitted transition is to INVALID.<br><br>This state is entered from TEST_DEV for special use cases in which RMA is not allowed. |
| RMA_SIP | 4'b001X | Enables the SiP owner to debug and test the part.<br><br>The secret token (RMA_SIP_TOKEN) must be presented to enable the transition from PROD or TEST_DEV. |
| RMA_CHIPLET | 4'b011X | Enables the chiplet owner to debug and test the part. All features are enabled (neither SIP_DIS nor SYS_DIS applies), unlike TEST_DEV, which still honors SIP_DIS/SYS_DIS.<br><br>The secret token (RMA_CHIPLET_TOKEN) must be presented to enable the transition from RMA_SIP. |
| INVALID | All other values | Disables all features unless SEC_DIS overrides it. Also used before fuse sensing completes; observing this profile alone does not establish that the OTP has been programmed to an end-of-life state. |

### Multiple Security Domains

The SiP owner controls the debug-access network between chiplets. Each chiplet owner controls access to its own local assets. A system has one shared SiP-owner domain and a local domain for each chiplet with independently protected assets.

`SIP_DBG` admits and forwards external debug traffic at each chiplet with a local SEP/LCC. Access to local resources additionally requires `CHIPLET_DBG`; SEP resources also require `SEP_DBG`. Opening another chiplet's local debug gate is unnecessary when traffic only passes through it.

A chiplet without a local SEP/LCC contributes no local disable vector. The preceding LCC-bearing chiplet controls forwarding to it, as described in Non-Primary Chiplets.

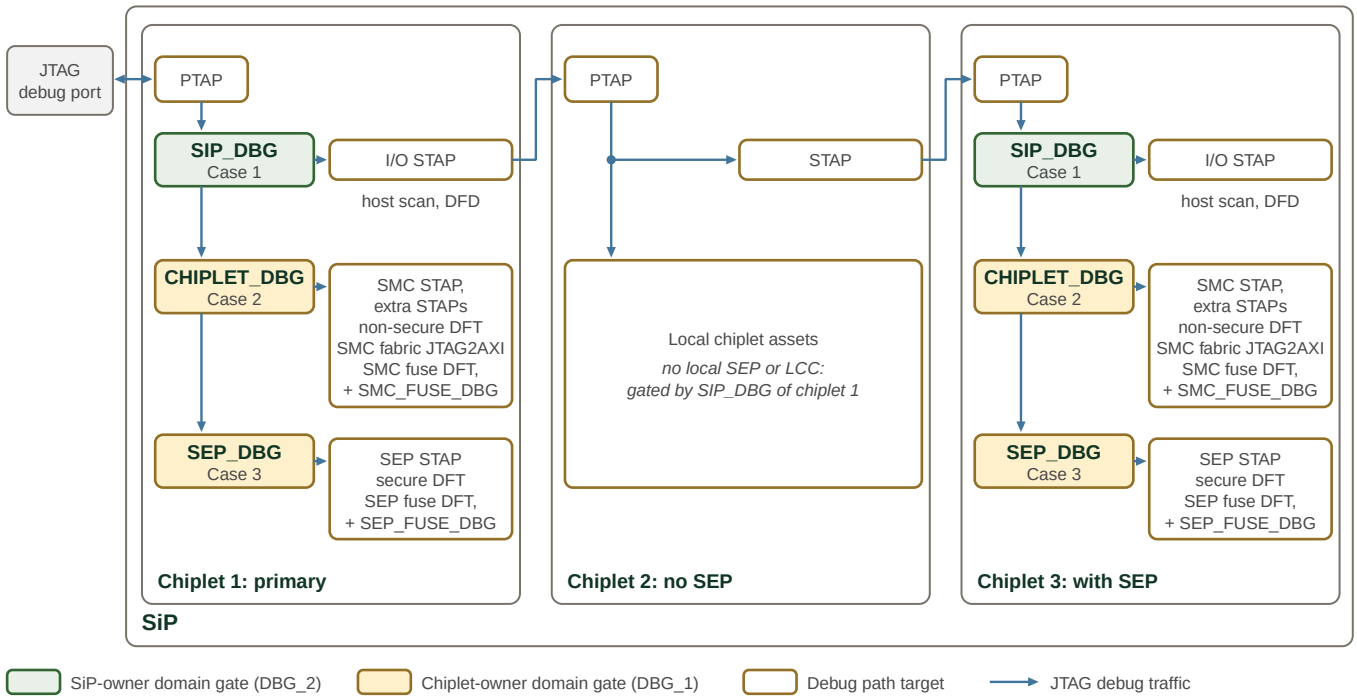


*Figure 30. SEP security domains*

In Figure 30, each chiplet with a SEP gates the debug path from its PTAP downwards: SIP_DBG opens the Case 1 targets, CHIPLET_DBG adds the Case 2 targets and SEP_DBG adds the Case 3 targets, each listed to the right of its gate. The targets are the debug paths that `sep_lifecycle_ctrl` disables through `dbg_disable_o` and the two fuse DFT disables, as given in Debug and Test Port Path Gating; the fuse DFT paths also need their own SEP_FUSE_DBG or SMC_FUSE_DBG bit. The green gate belongs to the SiP-owner domain and is a DBG_2 bit, opened in the field by Demotion 2. The cream gates belong to the chiplet-owner domain and are DBG_1 bits, opened by Demotion 1. The I/O STAP forwards the chain to the next chiplet's PTAP.

In Figure 30, chiplet 1 is primary, chiplet 2 has no SEP, and chiplet 3 has its own LCC:

- Chiplet 1 local debug requires `SIP_DBG(1)` and `CHIPLET_DBG(1)`.
- Chiplet 2 access requires `SIP_DBG(1)`; chiplet 2 adds no local gate.
- Chiplet 3 local debug requires `SIP_DBG(1)`, `SIP_DBG(3)` and `CHIPLET_DBG(3)`. It does not require `CHIPLET_DBG(1)`.

The per-path conjunctions are listed in Debug and Test Port Path Gating.

### Debug and Functional Feature Control

`feat_ctrl_o` uses positive logic: 1 enables a feature. The 64-bit `SIP_DIS` and `SYS_DIS` shadows use negative logic: 1 restricts a feature. The applicable masks depend on lifecycle state and demotion. Where both apply, the enable is `~(SIP_DIS | SYS_DIS)`.

Table 177 gives the complete state-dependent formulas. The final `SEC_DIS` override is described in Overrides and Final Gates (SEC_DIS).

### Disable Vector Format (SIP_DIS / SYS_DIS)

Table 175 defines the **disable** vector format — the bit layout shared by both SIP_DIS and SYS_DIS (each is 64 bits). Each defined bit gates a specific feature or its associated scan chain; the remaining bits are reserved for adopter extension

(see Asset and Scan-Chain Protection and Adopter Extension).

In the policy tables, `T` means enabled and `F` means disabled. A clear disable bit permits a feature only where the lifecycle profile allows it; it does not by itself open a production debug path.

DBG_1 covers `[23:0]` and is controlled by Demotion 1. DBG_2 covers `[47:24]` and is controlled by Demotion 2. Function bits occupy `[63:48]`.

*Table 175. SEP feature-disable vector*

| DIS bit index | Feature Category | Feature Name | Feature Description (`1'b1` disables the feature; `1'b0` enables it) |
|---|---|---|---|
| 0 | debug (DBG_1) | SEP_DBG | Coarse-grained debug enable for the **SEP** scope: the local gate for SEP internal state. `1'b1`: SEP-scope debug / scan-dump access disabled and inbound (`ext`) traffic is filtered before it can reach SEP internal state. `1'b0`: enabled — inbound traffic bypasses the filter and scan dump is allowed. SEP_DBG is a local gate, not the chiplet ingress: `SIP_DBG` (bit 24) admits external debug traffic into the chiplet, `CHIPLET_DBG` (bit 1) admits it to chiplet-internal resources, and `SEP_DBG` then governs whether it can reach SEP internal state. Debugging SEP assets from an external debugger therefore requires **all three** — Case 3 of Debug and Test Port Path Gating — not `SEP_DBG` alone. |
| 1 | debug (DBG_1) | CHIPLET_DBG | Coarse-grained debug enable for the **chiplet** scope. `1'b1`: chiplet-scope debug / scan-dump access disabled. `1'b0`: enabled. Never acts alone: an external debugger reaches chiplet-internal resources only when `SIP_DBG` is open as well (Case 2 of Debug and Test Port Path Gating). Contributes to the reachability of the crypto / KM secure chain but never authorizes a dump from it: that chain admits only test commands, and only while `secure_tm = 1`. In `TEST_DEV` and `RMA_SiP`, this bit set in `SIP_DIS` or `SYS_DIS` makes the production ROM enforce secure boot; it has no such effect in `RMA_CHIPLET`. See Firmware Authorization. |
| 2 | debug (DBG_1) | SEP_FUSE_DBG | `1'b1`: the DFT-inserted access path into the SEP OTP is disabled. `1'b0`: enabled. This gate also covers the OTP IP-vendor test and calibration interface (used during ATE for vendor-specific OTP test functions and internal-voltage calibration), which is reachable only over that inserted path. The functional JTAG2AXIL path to the OTP is always available and is not gated by this bit. |
| 3 | debug (DBG_1) | SMC_FUSE_DBG | `1'b1`: the DFT-inserted access path into the SMC OTP is disabled. `1'b0`: enabled. (The bit is produced by SEP-LCC's `feat_ctrl_o`, exported on the SEP boundary, and consumed by SMC to gate its own inserted fuse path.) |
| 4 | debug (DBG_1) | ENTROPY_DBG | Access to the entropy source's readable FIFOs (`FIFO_RDATA`, `BIW_OBS_RDATA`, `NOISE_OBS_RDATA`), which hold the DRBG seed words and pre-conditioning entropy. `1'b1`: disabled for every initiator, the SEP CPU included. `1'b0`: enabled. Adds no term to Debug and Test Port Path Gating and does not gate the entropy pool. See FIFO Access Gate. *Specified ahead of the hardware: until the RTL implements it, the FIFOs are accessible in every state.* |
| 5-23 | debug (DBG_1) | *Reserved* | Available for adopter-defined local (DBG_1) debug scan chains. See Adopter Extension. |
| 24 | debug (DBG_2) | SIP_DBG | Coarse-grained debug enable for the **SiP** scope: the inter-chiplet debug-bypass gate (DBG_2, gated by Demotion 2) that admits external (PTAP) debug traffic into this chiplet and forwards it onward to the next chiplet via the outbound JTAG PTAP. Debugging a chiplet's assets requires SIP_DBG enabled on each LCC-bearing chiplet along the debug path, including the target itself when it has its own LCC; chiplets without a local SEP/LCC contribute no gate (see Multiple Security Domains). `1'b1`: disabled. `1'b0`: enabled. |
| 25-47 | debug (DBG_2) | *Reserved* | Available for adopter-defined inter-chiplet debug / DFT forwarding paths. Each allocated bit is consumed on the LCC-bearing chiplet to gate the corresponding traffic before it is forwarded downstream. See Adopter Extension. |
| 48-63 | function | *Reserved* | Available for adopter-defined functional feature gating. See Adopter Extension. |

### DIS Vector Ownership: Predecessor-Programs-Disables

The vector name identifies the receiving owner, not the programmer:

- The chiplet creator programs `SIP_DIS` before SiP handoff. Its restrictions remain effective in RMA_SIP and are bypassed in RMA_CHIPLET.
- The SiP owner programs `SYS_DIS` before system handoff. Its restrictions are bypassed in RMA_SIP and RMA_CHIPLET.

RMA bypass does not erase either fuse vector. The system owner has no successor and therefore no third disable vector. It provisions its own `SYS_PUBK_HASH` and `SYS_UID` before TEST_DEV-to-PROD transition; see the OTP Fuse Controller chapter for the provisioning sequence.

#### Combining Rule: OR then Invert

Where both owner masks apply, a feature is disabled if either mask has its bit set. OR combines the restrictions; inversion produces the positive-logic enable. Since an OTP bit can only change from 0 to 1, a later owner cannot erase an earlier owner's programmed restriction. Lifecycle profiles can bypass masks for RMA access without changing OTP.

#### When Each DIS Vector Applies

The table below gives mask applicability before demotion and the final overrides. In TEST_DEV, demotion can bypass the masks for its debug group; in PROD, demotion opens that group only subject to both masks. See Table 177 for those exceptions.

Table 176 identifies which disable vectors apply in each lifecycle state.

*Table 176. SEP disable-vector applicability*

| LC state | Applicable DIS vectors |
|---|---|
| TEST_DEV | Both SIP_DIS and SYS_DIS apply. TEST_DEV is the shared phase the part passes through under the chiplet creator, SiP owner, and system owner in turn. |
| PROD / PROD_END | Both SIP_DIS and SYS_DIS apply. |
| RMA_SIP | Only SIP_DIS applies. At the SiP-owner level, the SiP owner sits **above** the system owner, so its own SYS_DIS — which restricts the system owner — does not bind it; only the chiplet creator's SIP_DIS restriction remains in force. |
| RMA_CHIPLET | Neither SIP_DIS nor SYS_DIS apply (all features enabled). |
| INVALID | Neither applies (all features disabled by fail-closed default). |
| (any) with SEC_DIS = 1 | Neither DIS vector applies (all features enabled). |

#### Overrides and Final Gates (SEC_DIS)

Two controls sit **on top of** the per-LC-state / DIS-vector computation, applied in this order (a later step overrides an earlier one):

1. **Signal-integrity fail-closed (base):** if the differentially encoded LC_STATE fails its integrity check (`lc_sigint_err`), the base `feat_ctrl` is forced to all-0s (all features disabled).
2. **SEC_DIS override:** when `SEC_DIS = 1`, `feat_ctrl` is forced to all-1s — every feature enabled, regardless of the LC state, the DIS vectors, or the sigint result (an emergency bring-up bypass on A0 silicon; see the Security Disable section).

`SECURE_TM` is **not** a term in `feat_ctrl_o`. No feature control bit is qualified by secure test mode: the debug group `[47:0]` and the function group `[63:48]` are both determined solely by the LC state, the DIS vectors, and the SEC_DIS override. `secure_tm` acts on the scan path and on the fuse controller rather than on feature control — see `secure_tm` and the Secure Scan Chain.

### Per-LC-state Feature Control Profile

The output is shown as DBG_1 / DBG_2 / Function. `T` enables all bits of a group; `F` disables them. A formula applies the indicated disable masks. Demotion has no effect on Function.

*Table 177. Per-LC-state feature control profile*

| LC_STATE | SEC_DIS | DEMOTE (1 / 2) | Feature Control: \{DBG_1 / DBG_2 / Function} | Description |
|---|---|---|---|---|
| TEST_DEV (4'b0000) | 0 | 0 / 0 | ~(SIP_DIS.DBG_1 \| SYS_DIS.DBG_1) / ~(SIP_DIS.DBG_2 \| SYS_DIS.DBG_2) / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | ATE/SLT test, fuse provisioning (SIP_DIS / SYS_DIS and the RMA tokens), and product firmware development. Both SIP_DIS and SYS_DIS apply; a demotion forces its debug group open, overriding the DIS vectors. Function is not affected by demotion. |
| | 0 | 1 / 0 | T / ~(SIP_DIS.DBG_2 \| SYS_DIS.DBG_2) / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 0 | 0 / 1 | ~(SIP_DIS.DBG_1 \| SYS_DIS.DBG_1) / T / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 0 | 1 / 1 | T / T / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 1 | X | T / T / T | |
| PROD (4'b0001) | 0 | 0 / 0 | F / F / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | Product state. A demotion relaxes its debug group to honor the DIS vectors (one level down from disabled). Both SIP_DIS and SYS_DIS still apply; Function is not affected by demotion. |
| | 0 | 1 / 0 | ~(SIP_DIS.DBG_1 \| SYS_DIS.DBG_1) / F / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 0 | 0 / 1 | F / ~(SIP_DIS.DBG_2 \| SYS_DIS.DBG_2) / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 0 | 1 / 1 | ~(SIP_DIS.DBG_1 \| SYS_DIS.DBG_1) / ~(SIP_DIS.DBG_2 \| SYS_DIS.DBG_2) / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | |
| | 1 | X | T / T / T | |
| PROD_END (4'b1000) | 0 | — | F / F / ~(SIP_DIS.FUNC \| SYS_DIS.FUNC) | Terminal product state — normal function only; the only permitted transition is to INVALID. Demotion has no effect. |
| | 1 | — | T / T / T | |
| RMA_SIP (4'b001X) | 0 | — | ~SIP_DIS.DBG_1 / ~SIP_DIS.DBG_2 / ~SIP_DIS.FUNC | SiP-owner debug/test. Honors SIP_DIS only (SYS_DIS does not bind the SiP owner). Entered via RMA_SIP_TOKEN. Demotion has no effect. |
| | 1 | — | T / T / T | |
| RMA_CHIPLET (4'b011X) | X | — | T / T / T | Chiplet-owner debug/test — all features enabled. Entered via RMA_CHIPLET_TOKEN. |
| INVALID (others) | 0 | — | F / F / F | The chip is not live: end of life, immediately after reset, or an invalid LC_STATE. |
| | 1 | — | T / T / T | |

In the DEMOTE column, a dash means the state ignores demotion; X means SEC_DIS makes its value irrelevant. The two demotions act independently.

### In-Field Debug Demotion

`DEMOTE_1` and `DEMOTE_2` each contain a `demote` bit and a `lock` bit. They are two registers, not separate demotion and lock register banks. Cold reset clears both and removes the volatile debug profile.

*Table 178. SEP demotion controls*

| Register | Feature group | Scope |
|---|---|---|
| `DEMOTE_1` | DBG_1, bits `[23:0]` | Local debug, including SEP and chiplet gates. |

| Register | Feature group | Scope |
|---|---|---|
| DEMOTE_2 | DBG_2, bits [47:24] | Inter-chiplet debug access and forwarding, including SIP_DBG. |

The controls are independent: either may be set first or used alone. In TEST_DEV, demotion forces its group open; in PROD, it enables the group subject to the disable vectors. Other lifecycle profiles and function bits are unchanged. The complete formulas are in Table 177.

Opening local debug does not open the external access path. For example, a Case 3 SEP debug path requires both groups; see Debug and Test Port Path Gating. The LCC applies register values without authenticating the writer or a manifest. Firmware authorization is discussed under Firmware Authorization.

### Programming and Locking

Both demote (bit 0) and lock (bit 1) are write-one-to-set fields. Writing zero does not clear them. While lock is zero, firmware can set demote; once locked, the demotion decision cannot change until cold reset. The lock can commit either a demoted or a non-demoted state. The demotion sequence is in the Programmer's Guide.

This is the hardware mechanism. The current ROM's manifest handling is documented in the Boot ROM chapter; the multi-owner authorization model above remains deferred.

### Reset Behavior

SEP cold reset clears the demote and lock fields in both registers. The LCC then applies the profile of the sensed OTP state without demotion.

### Asset and Scan-Chain Protection

Two distinct DFT operations use the scan chains, and the protection model treats them differently:

- **Scan test** — shift in a test pattern, run the functional clock for one or two cycles, then shift the result out. Used for manufacturing / DFT coverage.
- **Scan dump** — shift the live internal state out directly. Used for debug.

The integration must preserve test coverage without exposing live secrets. The rules below define the required scan partitioning and command gates; feature-vector generation alone does not implement or verify DFT insertion.

### Asset Classes

SEP assets are partitioned into three classes. Class 1 (non-scannable) and Class 2 (scannable for test coverage but **never** scan-dumpable) are each split by protection rationale — **a** for confidentiality and **b** for integrity. Class 3 is everything else.

Table 179 relates the asset classes to scan and scan-dump treatment.

*Table 179. SEP security asset classes*

| Class | Asset Examples | Scannable (test) | Scan-dumpable (debug) | Gate & Scan-Chain Treatment |
|---|---|---|---|---|
| **1a** (non-scannable; confidentiality) | UIDs (CHIPLET_UID, SIP_UID, SYS_UID); *_TOKEN_I JTAG input registers (SEC_DIS_TOKEN_I, RMA_SIP_TOKEN_I, RMA_CHIPLET_TOKEN_I); CLASS_KEY; raw fuse read-back data. | No | No | **Excluded** from all scan chains. **Confidentiality** rationale: these registers hold the actual secrets; the data lives **in** them, so the only safe option is to keep them off every scan chain. |

| Class | Asset Examples | Scannable (test) | Scan-dumpable (debug) | Gate & Scan-Chain Treatment |
|---|---|---|---|---|
| **1b** (non-scannable; integrity) | `LOCKS`/`LOCKS_SPARE`, `LC_STATE`, `SIP_DIS`, `SYS_DIS` shadow registers; `*_TOKEN_DIGEST` shadow registers; token-hash digest/valid and RMA match state | No | No | **Excluded** from all scan chains. **Integrity** rationale: the values are not secret (digests are SHA-256 preimage-resistant), but scan shifting could change them and illegitimately enable a disabled feature or make an attacker-presented token match. Stability against scan-induced perturbation is required. |
| **2a** (secret-bearing; confidentiality) | The complete Key Manager; AES, HMAC-SHA2, KMAC-SHA3, OTBN, entropy, DRBG/EDN, Adams Bridge, and TRNG sequential state; KM/OTBN companion SRAMs (see Class 2 Asset Manifest and Naming) | Yes | **Never** | **Dedicated secure scan chain.** Reachable when its SIB is opened by the `_DBG feature control bits; a *test` command is then admitted only when `secure_tm = 1`, and at `secure_tm = 0` no command is admitted at all. The secret-bearing shadow registers (UIDs / `CLASS_KEY`) are disconnected from downstream while `secure_tm = 1`, so these consumers hold no real secret during scan test. **Scan dump is never permitted — for any owner or LC state**: a dump command is never admitted on this chain (see `secure_tm` and the Secure Scan Chain). |
| **2b** (fuse program path; integrity) | Fuse programming **control** MMRs and **status** MMRs of the SEP fuse controller | Yes | **Never** | **Dedicated secure scan chain**, alongside Class 2a, under the same command-class rule. **Integrity** rationale: shifting is not read-only with respect to the flops it passes through. During a dump every flop in the chain takes on a succession of arbitrary transient values, so a program-control register on a dumpable chain would pass through the encoding for "program" during an ordinary dump, and the dump itself could burn fuses. Status registers are included because spoofed status can steer the program sequencer. Class 2 rather than Class 1 so the fuse controller keeps manufacturing defect coverage; at `secure_tm = 1` the OTP write-enable inputs are tied inactive, so scan toggling cannot reach the fuse array. |
| **3** (general logic) | The rest of the logic not classified above | Yes | Yes (gated) | **General scan chains.** Scannable and scan-dumpable when permitted by the `*_DBG` feature control bits (the scope's local gate — `SEP_DBG` or `CHIPLET_DBG` — plus the `SIP_DBG` debug-access path; see Multiple Security Domains) and the LC state. `secure_tm` plays no part here: these chains hold no secret, so both command classes are admitted whenever the chain is reachable. |

### `secure_tm` and the Secure Scan Chain

`secure_tm` is the secure-test-mode indicator: the `TEST_EN` strap latched into the internal `SECURE_TM` signal. It is latched when SEP fuse sensing completes if `SEC_DIS` is not asserted, or on chiplet cold-reset release if `SEC_DIS` is asserted (see the Test Mode section).

`secure_tm` establishes the required test configuration; it does not identify or authenticate the owner at the JTAG port. The protection rules require it to:

1. Admit test commands on the secure chain only while asserted.
2. Block updates to `LOCKS`, `LC_STATE`, `SIP_DIS` and `SYS_DIS` from SEP, OTP, JTAG and TRoot while the part is shiftable.
3. Hold OTP write-enable inputs inactive.
4. Disconnect UID and `CLASS_KEY` shadow outputs from downstream consumers.

Secure-chain access requires both an open SIB, under the feature gates in Debug and Test Port Path Gating, and an allowed command. Test commands require `secure_tm = 1`; dump commands are never allowed, for any owner or lifecycle state. With `secure_tm = 0`, no command is admitted on that chain.

These conditions distinguish Class 1 from Class 2. Class 1 registers contain secrets or state that directly controls protection, so they are excluded from scan. Class 2 consumers and fuse-program controls may be scanned only with their secret sources disconnected and the fuse-write path blocked. The feature bits control reachability; they do not waive these command rules.

The adopter's DFT implementation must enforce this separation. General Class 3 chains follow their debug gates and do not use `secure_tm` as an additional gate.

#### Feature-Bit to Scan-Chain Mapping

Each **general** (Class 3) scan chain is gated by a per-scope `*_DBG` feature control bit in `feat_ctrl_o`; the bit's owner — which predecessor DIS vector sets its default (see DIS Vector Ownership: Predecessor-Programs-Disables) — sets the multi-owner default policy. External access to any general chain additionally requires the `SIP_DBG` debug-access path into the chiplet to be open (see Multiple Security Domains). The Class 2 secure chain is reached through its own SIB, which the feature control bits gate in the same way; what distinguishes it is that reaching it never permits a dump, because the DFT controller admits only test commands there and only while `secure_tm = 1`.

Table 180 maps each asset class to its scan chain and dump gate.

*Table 180. SEP scan-chain gates*

| Asset Class | Scan Chain | Dump Gate | Notes |
|---|---|---|---|
| 1a / 1b (non-scannable) | — | — | Excluded from all scan chains. |
| 2a (secret-bearing) + 2b (fuse program path) | Shared dedicated secure chain | Command class — a dump command is never admitted; a test command requires `secure_tm = 1` | Never scan-dumpable, any owner / LC state. Its SIB is feature-bit gated like any other chain, but opening it grants test access only — with the shadow secrets disconnected (2a) and the OTP write-enables tied inactive (2b). |
| 3 (general logic) | Per-scope general chains | Local scope `*_DBG` bit (`SEP_DBG`, `CHIPLET_DBG`) + `SIP_DBG` access path + LC state | Coarse-grained: a scope's general logic is scan-dumpable only when its local `*_DBG` bit enables it, the `SIP_DBG` debug-access path is open, and the LC state permits. |

Reserved-bit ranges available for adopter-defined scan chains (see Adopter Extension):

- Debug group: bits 5-23 (DBG_1, 19 reserved bits) and bits 25-47 (DBG_2, 23 reserved bits)
- Function group: bits 48-63 (16 reserved bits)

#### Debug and Test Port Path Gating

DTP reachability uses three nested combinations of feature bits. A granular fuse-debug bit adds another AND term; it never replaces the case's required bits.

Table 181 defines the three cases used to determine debug-path access.

*Table 181. DTP access cases*

| Case | Condition | Applies to |
|---|---|---|
| 1 | `SIP_DBG` | Paths that only enter or traverse the chiplet without reaching chiplet-internal resources. |
| 2 | `SIP_DBG & CHIPLET_DBG` | Paths reaching chiplet-internal resources. |
| 3 | `SIP_DBG & CHIPLET_DBG & SEP_DBG` | Paths reaching the SEP or the secure domain. |

Case 1 admits traffic, Case 2 reaches chiplet resources, and Case 3 reaches SEP resources. Every bit in the applicable case

must be enabled. For example, leaving SEP_DBG open while CHIPLET_DBG is closed does not grant SEP access.

Table 182 lists the feature-control terms required for each DTP path.

*Table 182. DTP path feature gates*

| DTP path | Gating | Notes |
|---|---|---|
| I/O S-TAP host path | Case 1 | |
| External S-TAP return path | Case 1 | Onward forwarding to a downstream chiplet is part of the SiP-owner domain (see Multiple Security Domains). |
| DFD iJTAG path | Case 1 | |
| SMC debug S-TAP host path | Case 2 | No dedicated SMC debug bit exists; see the note below. |
| DTP configurable (extra) S-TAP host paths | Case 2 | |
| Non-secure external DFT iJTAG path | Case 2 | The DFT non-secure chain SIB. Not gated by `secure_tm` — see `secure_tm` and the Secure Scan Chain. |
| SMC fabric JTAG2AXI path | Case 2 | |
| SEP debug S-TAP host path | Case 3 | Requires the full Case 3 conjunction; `SEP_DBG` alone is not sufficient. |
| Secure external DFT iJTAG path | Case 3 | The DFT secure chain SIB. Reaching it grants test access only, under the command-class rule of `secure_tm` and the Secure Scan Chain. |
| DFT-inserted SEP fuse path | Case 3 `& SEP_FUSE_DBG` | Includes the OTP IP-vendor test and calibration interface, which is reachable only over this path. |
| DFT-inserted SMC fuse path | Case 2 `& SMC_FUSE_DBG` | |
| SEP OTP JTAG2AXIL path | **Ungated** | Reaches the whole SEP eFuse interface: the shadow register map, the token-input MMRs, and the eFuse interface control registers, fuse program and read controls included. No feature control bit gates it. Enforcement is per-field `LOCKS` plus a wrapper access policy that, in PROD, RMA_SIP, or on an LC differential-decode integrity error, admits only the MMR region for both reads and writes, so an RMA token can still be presented. `LOCKS` is only in force once sensed, so this depends on nothing on this path being able to execute before fuse sensing completes. |
| SMC OTP JTAG2AXIL path | **Ungated** | Reaches the whole SMC eFuse interface on the same terms, under a stricter access policy. SMC holds no `LC_STATE` fuse field of its own; its wrapper decodes the state forwarded from SEP. In PROD or RMA_SIP it admits no writes at all and only reads of `CHIPLET_ID` and `PACKAGE_ID`; on an integrity error it admits nothing. |

Case 3 spans both demotion groups: SEP_DBG and CHIPLET_DBG are in DBG_1, while SIP_DBG is in DBG_2. In PROD, enabling only one demotion leaves a Case 3 path closed.

There is no separate SMC_DBG bit. SMC debug shares Case 2, so opening CHIPLET_DBG also admits SMC debug when SIP_DBG is open. A separate SMC policy would require an adopter-defined bit and corresponding path gating.

### Non-Primary Chiplets

Primary chiplets always include a local SEP and LCC; the asset partitioning described in Asset and Scan-Chain Protection applies directly. **Non-primary** chiplets may not include a local SEP, in which case debug access is governed by the **SiP-owner domain**. The primary chiplet's SiP-owner feature control gates the DFT traffic before forwarding it to the non-primary chiplet (see Multiple Security Domains).

For non-primary chiplets, the adopter is responsible for:

- Identifying which non-primary chiplet assets require protection.
- Partitioning those assets across scan chains analogous to the class scheme in Asset Classes (1a/1b, 2, 3).

- Routing the non-primary chiplet's scan / DFT traffic through the appropriate DTP path on the primary chiplet, where SiP-owner feature control gates forwarding.
- Applying the same scan-protection treatment to any non-primary chiplet logic that handles secrets: Class 1 excluded from scan chains, Class 2 never scan-dumpable.

The `SIP_DBG` mechanism documented in Disable Vector Format (SIP_DIS / SYS_DIS) gates the DTP path on the primary chiplet. The primary LCC therefore governs debug access to a non-primary chiplet by controlling whether DFT traffic is forwarded to it.

#### Adopter Extension

If an adopter integrates SEP into a chiplet with assets that require their own scan-chain protection — assets not covered by the chiplet-scope SEP scan chain or the general-logic scan chain in Asset and Scan-Chain Protection — the LCC architecture accommodates this via the **reserved bits** in `feat_ctrl_o` (and the matching positions in SIP_DIS / SYS_DIS):

- Debug group: bits 5-23 (DBG_1, 19 reserved bits) and bits 25-47 (DBG_2, 23 reserved bits)
- Function group: bits 48-63 (16 reserved bits)

OCAH assigns no standard function to these reserved bits. An adopter may allocate a DBG_2 bit to an additional inter-chiplet DFT forwarding path; the bit is consumed on the LCC-bearing chiplet to gate the corresponding traffic before it is forwarded downstream.

Each new scan chain or forwarding path is gated by exactly one reserved bit. The corresponding bit position in SIP_DIS or SYS_DIS (chosen by which predecessor owner sets the default policy — see DIS Vector Ownership: Predecessor-Programs-Disables) determines the default OTP-programmed restriction.

Constraints any adopter extension must satisfy:

- **One feature control bit per scan chain.**
- **Asset-class compliance.** Secret-bearing fields used by the new chain must follow the asset-class scheme (see Asset and Scan-Chain Protection): actual secrets and integrity-critical state are excluded from scan chains (Class 1). A Class 2 chain must never admit a dump command. It may admit test commands only while `secure_tm = 1`, with secret sources disconnected and fuse programming blocked as specified above. The chain's feature-control bit controls reachability; it does not authorize a dump.
- **Per-LC-state policy.** The new bit follows the standard per-LC-state formulas for its group (see Table 177). DBG_1 and DBG_2 are disabled in undemoted PROD and in PROD_END; their respective demotion controls relax PROD to honor the disable vectors. Function bits continue to honor `~(SIP_DIS.FUNC | SYS_DIS.FUNC)` in both production states and are unaffected by demotion. All groups follow their listed TEST_DEV and RMA policies and the SEC_DIS override.

| | |
|---|---|
| **NOTE** | The specific bit-allocation convention — which reserved bits are reserved for which adopters, and how bit assignments are advertised in the RDL / package outputs — is currently **open** and will be defined in a future revision of this specification. |

#### Firmware Authorization

The LCC does not inspect manifests or signatures. It supplies the lifecycle state, and the OTP Fuse Controller chapter defines the fuse fields that govern firmware authorization: `SBOOT_DIS`, the `SIP_DIS` and `SYS_DIS` debug vectors, the public-key digests, `CHIPLET_PUBK_REVOKE`, and the `REQUIRED_SIGNERS` and `REQUIRED_ALGS` fields reserved for a deferred multi-owner signing model. Firmware, not the LCC, interprets them. The SEP Boot ROM reads `CHIPLET_DBG` from the `SIP_DIS` and `SYS_DIS` fuse shadows rather than from `FEAT_CTRL`, because `SEC_DIS` and demotion can force `FEAT_CTRL` open on a part

whose fuses disable debug.

The Secure Boot Decision section of the SEP Boot ROM appendix in the Programmer's Guide specifies when the ROM requires a verified manifest, including the rule that a debug-locked `TEST_DEV` or `RMA_SiP` part is treated as `PROD`. The SiP Secure Boot application note summarizes that decision and describes the multi-owner model, which the current ROM does not implement, under deferred multi-owner signing model there.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### SEP Lifecycle Control Address Map

Lifecycle feature-control and demotion registers with their associated locks.

*Table 183. SEP Lifecycle Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | FEAT_CTRL | R | - |
| 0x8 | DEMOTE_1 | RW | - |
| 0x10 | DEMOTE_2 | RW | - |

#### FEAT_CTRL

*Table 184. sep_lifecycle_ctrl.FEAT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `feature_control` | R | 0x0 | Lifecycle feature control bits, |

#### DEMOTE_1

*Table 185. sep_lifecycle_ctrl.DEMOTE_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `rsvd` | RW1S | 0x0 | Not used |
| 1 | `lock` | RW1S | 0x0 | Demote lock bit. Once locked, demote can no longer be changed |
| 0 | `demote` | RW1S | 0x0 | Demote bit to enter PROD_DBG state |

#### DEMOTE_2

*Table 186. sep_lifecycle_ctrl.DEMOTE_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `rsvd` | RW1S | 0x0 | Not used |
| 1 | `lock` | RW1S | 0x0 | Demote lock bit. Once locked, demote can no longer be changed |
| 0 | `demote` | RW1S | 0x0 | Demote bit to enter PROD_DBG state |

#### Test Mode Entry

SECURE_TM selects the protected configuration required for secure scan tests. It does not authenticate a JTAG owner or enable debug access. Lifecycle state, SIP_DIS, SYS_DIS and the SEC_DIS override determine the debug and function bits of feat_ctrl_o; SECURE_TM does not qualify those bits.

#### Entry and Exit

Connect the chiplet's TEST_EN strap to the SEP wrapper's secure_tm_req_i. Hold it stable through reset release, fuse sensing and the test session. The entry contract requires the sampled SECURE_TM value to remain asserted until chiplet cold reset ends the session; software has no clear operation.

*Table 187. Secure test-mode sampling in the SEP wrapper*

| Condition | Action |
|---|---|
| Cold reset asserted | Clear SECURE_TM to 0. |
| SEC_DIS asserted at the early capture point | Sample the strap on the second rising clock edge after wrapper cold-reset release. |
| Fuse-sense completion rises | Sample the strap, including when SEC_DIS is asserted. |
| Neither sampling event | Retain the sampled value. |

The early capture counter saturates after that opportunity: asserting SEC_DIS later does not itself sample the strap. A subsequent fuse-sense completion can sample it again. The current wrapper loads the strap value at each sampling event; it is not a set-only latch. If the strap changes from 1 to 0 between captures, the later capture can clear SECURE_TM without reset. The stable-strap requirement is therefore necessary to meet the reset-only exit contract; that contract is not an unconditional hardware guarantee for a changing strap.

#### Protection While Active

The secure-test configuration must enforce all four protections below. The adopter's DFT integration remains responsible for secure-chain separation and command admission.

*Table 188. Required secure-test protections*

| Protected resource | Requirement while SECURE_TM=1 |
|---|---|
| Dedicated secure scan chain | Admit test commands only through an open, feature-gated scan path. Never admit dump commands. At SECURE_TM=0, admit no command on this chain. |
| Protection shadows | Block all updates to LOCKS, LOCKS_SPARE, LC_STATE, SIP_DIS and SYS_DIS. |
| OTP programming path | Hold the fuse macro's write-enable inputs inactive. |
| Secret outputs | Disconnect CHIPLET_UID, SIP_UID, SYS_UID and CLASS_KEY from downstream consumers. |

The eFuse controller implements request blocking and masks the designated secret shadow outputs to zero. Masking does not erase the stored fuse or shadow values, and register reads retain their normal access controls. It must not be treated as proof that every downstream consumer has been cleared. These protections allow testing without exposing persistent secrets that cannot be erased before RMA.

General scan chains follow their ordinary debug gates; the dedicated secure chain has the additional command restrictions above. For the normative scan classes and access rules, see secure_tm and the Secure Scan Chain.

### 2.1.5.11. Cryptographic Subsystem

The cryptographic accelerators use separate paths for data and secret keys:

- The SEP system fabric drives the AXI4 data and CSR bus.
- Key Manager drives the AXI4-Lite secret-key buses.

The accelerators are derived from OpenTitan, except for the Adams Bridge post-quantum accelerator from Caliptra. The firmware is based on the OpenTitan cryptographic library. Each block's reference page includes its available register maps; the Detailed Register Map links to those pages.

OCAH adds bus adapters and separate key-delivery paths to the accelerator wrappers. Key Manager writes keys through AXI4-Lite interfaces; the wrappers connect them to the accelerators' key inputs. The following sections describe the shared integration rules and accelerator-specific differences.

Adams Bridge requires 13 SRAM macros with masking enabled (default), or 9 with masking disabled.

Figure 31 shows the accelerator interconnect, Key Manager paths, and external memory interfaces.

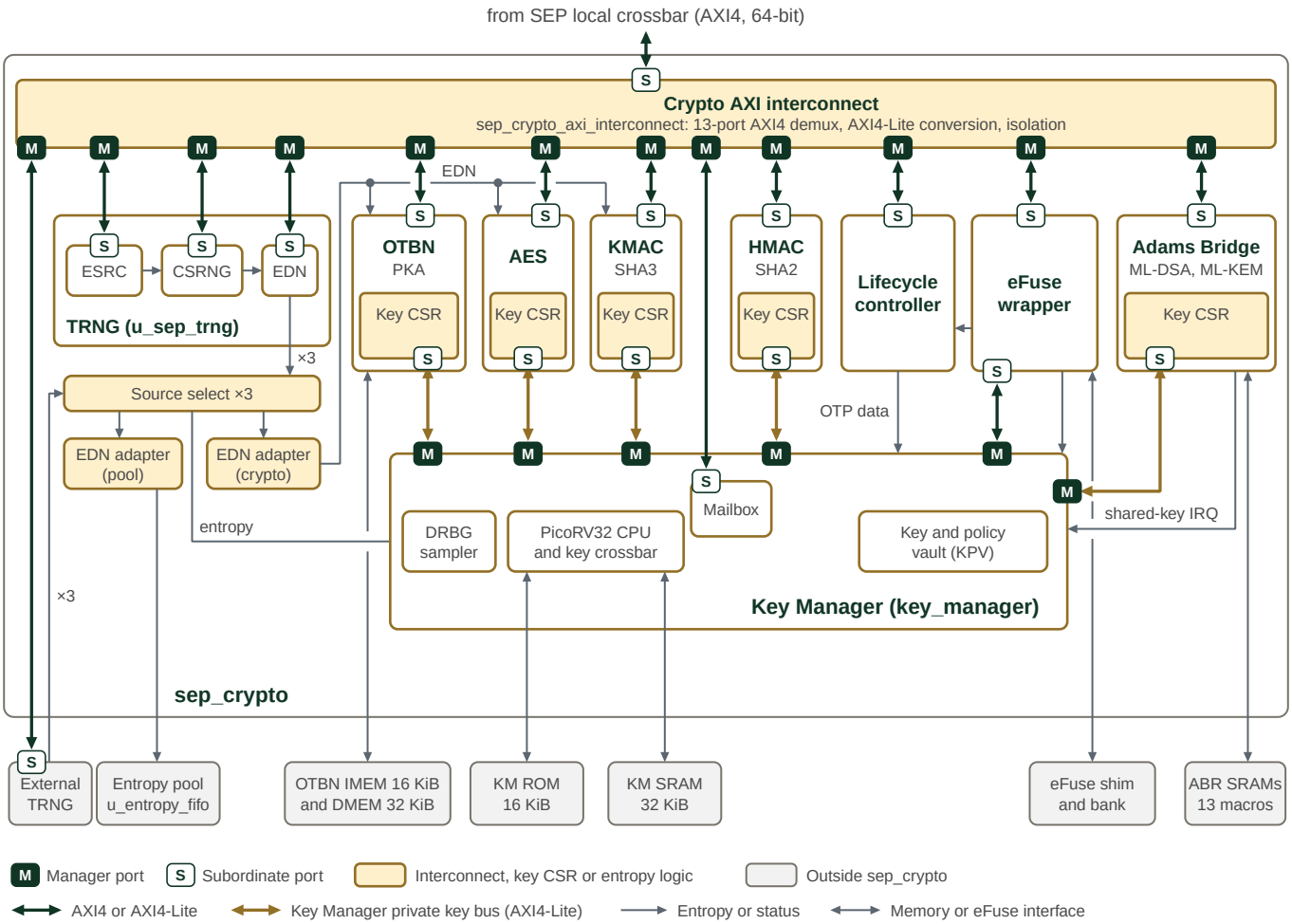


*Figure 31. SEP cryptographic subsystem and external memory interfaces*

In Figure 31, dark lines are host AXI4 and AXI4-Lite paths from the crypto AXI interconnect, and brown lines are the Key Manager private key buses. Each accelerator's key CSR block is the `<accelerator>_wrapper_key_reg` block that receives the `KEY_SHARE0`, `KEY_SHARE1` and `KEY_CTRL` writes described in Key Manager Key Delivery and drives the core's sideload key port. The Key Manager reaches the key CSR blocks through its internal crossbar; the Adams Bridge key bus leaves on the right, beside the ML-KEM shared-key interrupt that Adams Bridge returns. The host paths to the Key Manager mailbox, the external TRNG registers, the eFuse wrapper and the lifecycle controller are not isolated, and the eFuse and lifecycle legs leave the interconnect as AXI4. All six Key Manager paths, including its eFuse access, pass an isolation stage in

sep_crypto_axi_interconnect.

The Key Manager also receives OTP data: the eFuse shadow registers supply the lifecycle state and the identity and class-key fields, and the lifecycle controller supplies the two demotion states. Grey arrows carry entropy, OTP data and status one way; the three entropy streams from the DRBG pass through a source select that can take the external TRNG instead, as the TRNG and DRBG page describes. Grey boxes are outside sep_crypto: the memory macros live in the integration layer, and the entropy pool is u_entropy_fifo in sep. The Adams Bridge SRAMs are 13 macros with masking enabled and 9 without. Interrupts and OpenTitan alerts are not drawn; Table 189 lists them.

The integrator must instantiate the Adams Bridge working SRAMs outside SEP and connect them through the dedicated abr_mem request/response interface. For the memory interface and configuration requirements, see SEP Memory Interfaces in the Integrator Guide.

The sep_crypto module implements the integration layer, with sep_crypto_axi_interconnect centralizing its bus infrastructure. A single 64-bit AXI slave port from the SEP system fabric is decoded to the accelerators and internal peripherals. The host paths to OTBN, HMAC, AES, KMAC, ESRC, CSRNG, and EDN are converted to AXI4-Lite before isolation, with data-width conversion where required by the endpoint. The OpenTitan wrappers then bridge the isolated AXI4-Lite paths to TL-UL.

The Key Manager drives private AXI4-Lite key buses to the accelerator wrappers, which map key words into the sideload structures expected by the OpenTitan cores. These buses are independently isolated so that they drain before either the Key Manager or the destination accelerator enters software reset.

Each accelerator has its own software-controllable reset. The SEP Reset Controller drains and isolates both its host and Key Manager paths before asserting reset; a Key Manager reset similarly drains all of its key paths. The entropy source, CSRNG, and EDN share one coordinated reset. See SEP Reset Controller and Resetting the TRNG.

Figure 32 shows how the resets from the SEP Reset Controller reach the instances inside u_sep_crypto.

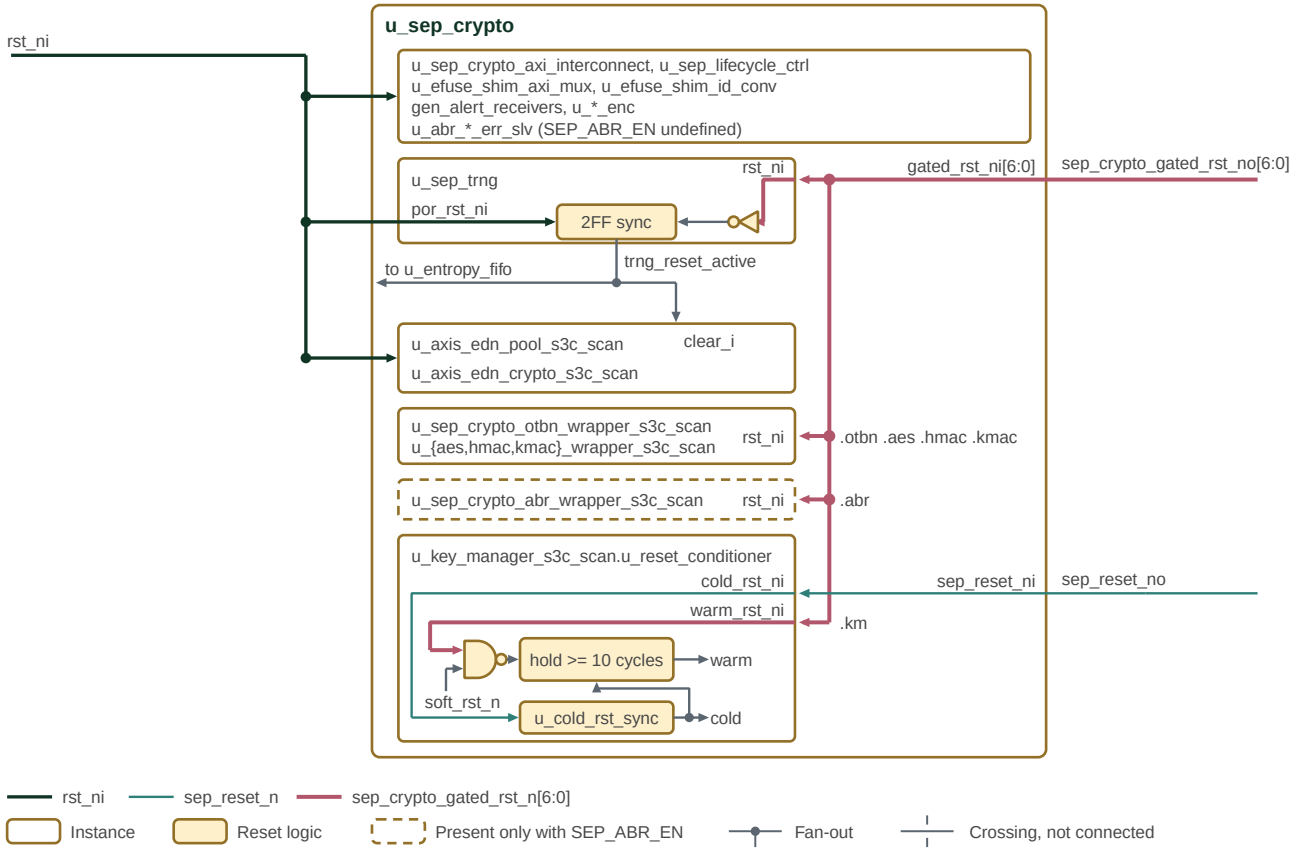


*Figure 32. SEP cryptographic subsystem resets*

Inside `u_sep_trng`, `por_rst_ni` resets a 2-flop synchronizer that samples the inverted TRNG reset and drives `trng_reset_active`. That signal synchronously clears both EDN AXIS adapters and, as `trng_entropy_clear_o`, the entropy pool. Both adapters take `rst_ni`. An OTBN, AES or KMAC software reset cancels that engine's endpoints in the crypto adapter (see SEP Reset Controller). The Key Manager conditions `sep_reset_n` into its cold domain through `u_cold_rst_sync`, and the gated Key Manager reset or its software reset `soft_rst_n` into a warm reset held for at least 10 cycles, as Figure 44 shows in more detail.

### OpenTitan Alert Termination

SEP does not instantiate OpenTitan's `alert_handler`. Each OpenTitan crypto alert channel is terminated locally by a `prim_alert_receiver` in `sep_crypto`. That primitive decodes the sender's differential `alert_p`/`alert_n` handshake and pulses `alert_o` when the handshake starts; a malformed encoding raises `integ_fail_o`. Alert ping is disabled (`ping_req_i` tied 0); `ping_ok_o` is unused.

PIC sources are in SEP CPU Interrupt Vector Map. A receiver's pulse and integrity-fail stay per-block at the receiver. Table 189 shows which of those, the native error IRQs, and the bridge faults are OR'd onto a PIC source and which are not.

*Table 189. SEP crypto alert and fault routing*

| Block | OT alert channels | Alert PIC | Native error IRQ | Bridge `err_o` |
|---|---|---|---|---|
| HMAC | `[0]` fatal | OR into Crypto alert | dedicated PIC | STATUS bit; OR into Peripheral register-bridge fault |
| OTBN | `[2:1]` | OR into Crypto alert | none | STATUS bit; OR into Peripheral register-bridge fault |

| Block | OT alert channels | Alert PIC | Native error IRQ | Bridge `err_o` |
|---|---|---|---|---|
| AES | `[4:3]` recoverable, fatal | OR into Crypto alert | none | STATUS bit; OR into Peripheral register-bridge fault |
| KMAC | `[6:5]` | OR into Crypto alert | dedicated PIC | STATUS bit; OR into Peripheral register-bridge fault |
| CSRNG | `[8:7]` | OR into Crypto alert | dedicated PIC | STATUS bit; OR into Peripheral register-bridge fault |
| EDN | `[10:9]` | OR into Crypto alert | dedicated PIC | STATUS bit; OR into Peripheral register-bridge fault |

Adams Bridge and the entropy source do not go through this conversion.

The Crypto alert source is the OR of every receiver pulse and integrity-fail, so that interrupt does not name the block. Native error IRQs are not OR'd. HMAC, AES, KMAC, OTBN, CSRNG, and EDN convert isolated AXI4-Lite to TL-UL through `axi_lite_to_tlul`; each bridge's sticky `err_o` is that wrapper's `bus_err_o`, visible as one bit in `PERIPH_BUS_ERR_STATUS` and cleared with the matching `PERIPH_BUS_ERR_CLEAR` bit. The Peripheral register-bridge fault source is the OR of those bits and the watchdog timer's bridge fault. Acknowledging one STATUS bit cannot discard another.

The crypto complex uses whole-instance Class 2 scan boundaries so all sequential key, round-key, digest-working, seed, and random-distribution state is covered without per-register marking. The marked roots in `sep_crypto` are `u_hmac_wrapper_s3c_scan`, `u_sep_crypto_otbn_wrapper_s3c_scan`, `u_aes_wrapper_s3c_scan`, `u_kmac_wrapper_s3c_scan`, `u_sep_crypto_abr_wrapper_s3c_scan`, `u_entropy_source_s3c_scan`, `u_drbg_s3c_scan`, `u_axis_edn_crypto_s3c_scan`, `u_axis_edn_pool_s3c_scan`, and `u_key_manager_s3c_scan`. The integration module itself, public bus infrastructure, and combinational entropy muxes are not Class 2 roots.

### Single-Beat Access Only

The `sep_crypto` demux enforces single-beat access to the crypto region (Single-Beat Register Access). Any access whose `AxLEN` is non-zero is routed to the internal error slave and answered with DECERR on every beat, whatever its address; no beat reaches an accelerator.

### Key Manager Key Delivery

The Key Manager delivers keys and seeds to the crypto accelerators over a private key interface that software cannot read. Every delivery uses the same format: the value is carried as two shares whose XOR is the value, and each share is presented as a sequence of 32-bit words in the `KEY_SHARE0` and `KEY_SHARE1` registers of the destination. Within a share, word index N carries value bits `[32N+31:32N]`, so index 0 is the least-significant word and the highest index is the most-significant.

Both shares are always consumed. Delivering a value V therefore requires two shares whose XOR is V; writing a single share delivers V only because the unwritten share reads as zero. `KEY_CTRL.KEY_VALID` qualifies the delivered value, and while it is 0 the destination has no valid key or seed.

The words per share, whether software must select the delivered key in preference to the accelerator's own software key registers, whether software must also declare the key length, and the behavior on an invalid or mismatched key are accelerator-specific. They are stated in the accelerator reference pages and summarized in Table 190; the delivered-key section of the Programmer's Guide gives the corresponding software configuration.

*Table 190. Key Manager key delivery per accelerator*

| Accelerator | Words per share | Select delivered key | Key length | Invalid or mismatched key |
|---|---|---|---|---|
| HMAC | 8 (256-bit) | Not required | `CFG.key_length`, set by software | Digest is wrong but repeatable; no error |

| Accelerator | Words per share | Select delivered key | Key length | Invalid or mismatched key |
|---|---|---|---|---|
| AES | 8 (256-bit) | `CTRL_SHADOWED.sideload` | `CTRL_SHADOWED.key_len`, set by software | Delivered key ignored while the select bit is 0; no error |
| KMAC | 8 (256-bit) | `CFG_SHADOWED.sideload` | Applied automatically | Keyed operation raises an error |
| OTBN | 12 (384-bit) | Not required | Not applicable | `ERR_BITS.KEY_INVALID`; the program stops |
| ABR | 8 per block (256-bit) | Seed read enable | Not applicable | Seed read does not complete |

### TRNG and DRBG

SEP separates physical entropy generation from deterministic random-data production and distribution. The internal path is:

- Entropy Source (ESRC): ring-oscillator sampling, extraction, health checks and SHA-256 conditioning.
- Cryptographically Secure Random Number Generator (CSRNG): an AES-CTR deterministic random bit generator (DRBG), seeded by ESRC.
- Entropy Distribution Network (EDN): requests generated data from CSRNG and supplies the three SEP sink paths below.

Each sink can instead select an external stream. The source muxes sit after EDN, so external data bypasses the internal ESRC/CSRNG/EDN chain. Stream availability and FIPS tags do not establish entropy quality or certification.

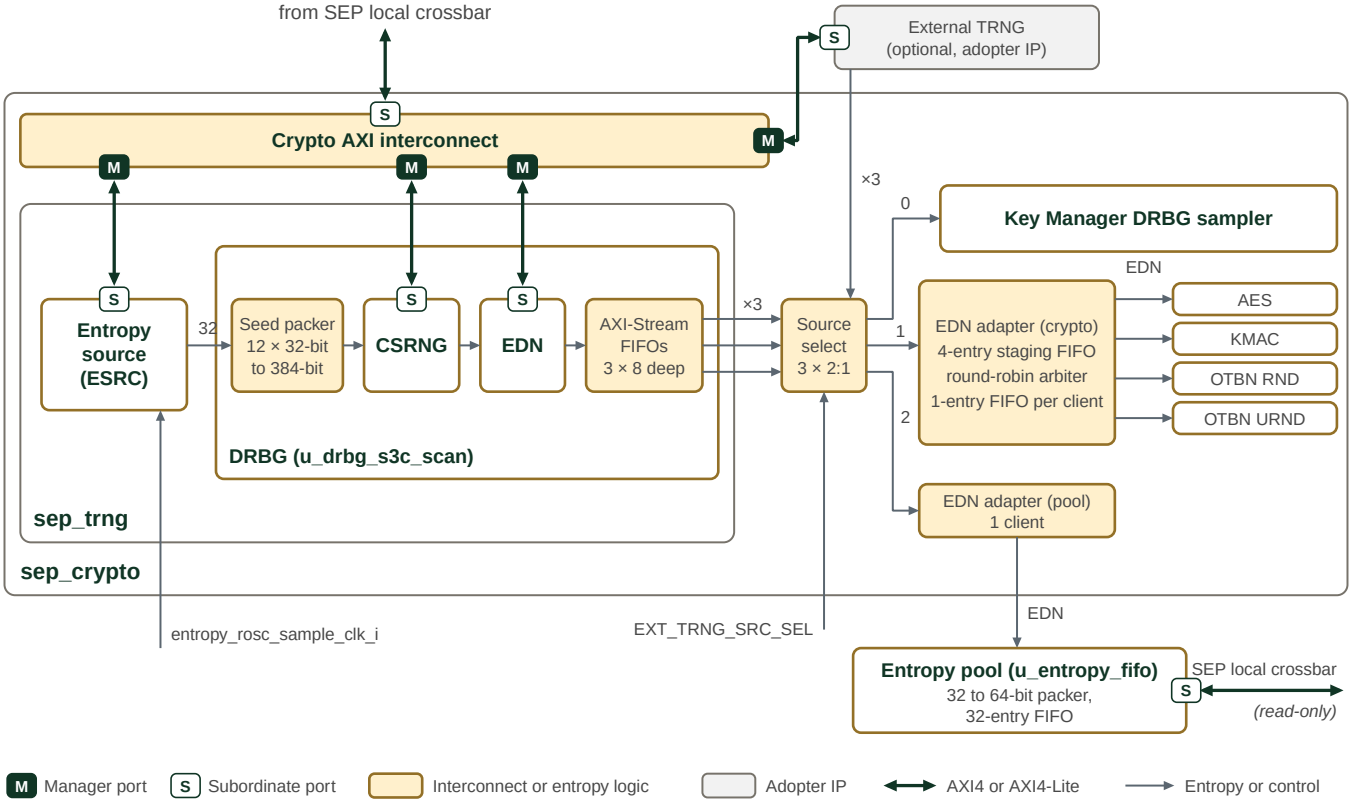


*Figure 33. SEP TRNG and DRBG entropy distribution*

Figure 33 reads left to right. Inside `u_sep_trng`, the entropy source drives a 32-bit stream into the DRBG, whose seed packer collects twelve words into each 384-bit CSRNG seed. EDN draws from CSRNG and fills one 8-entry AXI-Stream FIFO per endpoint. The crypto AXI interconnect reaches the entropy source, CSRNG and EDN registers, and the external TRNG register window. The source select holds one 2:1 mux per stream; each bit of `EXT_TRNG_SRC_SEL` chooses the

internal DRBG (0) or the external TRNG (1, the reset value) for its stream. Streams 0, 1 and 2 go to the Key Manager DRBG sampler, to the crypto EDN adapter and to the pool EDN adapter. The entropy pool sits in `sep`, outside `sep_crypto`, and packs the 32-bit words into a 32-entry FIFO of 64-bit words that software reads through the SEP local crossbar. The CSRNG and EDN cores are OpenTitan IP; their internal structure and register semantics are described in the OpenTitan CSRNG and EDN documentation. The Programmer's Guide gives the programming guidance for this block.

### Entropy Generation

The internal ESRC samples ring oscillators, decorrelates/extracts their output and conditions it with SHA-256. Its gating health test observes the extracted 32-bit stream before SHA-256; per-generator health results provide additional monitoring. The health tests include repetition, adaptive-proportion and Markov checks. This is not a test of each DRBG output word.

Startup health testing gates release into the normal output path. The stream to DRBG is the ESRC output selected by its configuration: normally SHA-256 output, but compressor-bypass mode selects uncompressed generator data and disables the gating 32-bit health test. Do not treat bypass or diagnostic settings as equivalent to the qualified operating configuration.

Source characterization, health-test thresholds and permitted configurations belong to the entropy-source qualification. Conditioning does not by itself establish the amount of physical entropy in a seed. The source requirements are described by NIST SP 800-90B.

See Entropy Source for the source controls, startup status, health monitoring and configuration locks.

### Distribution and Source Selection

SEP instantiates `drbg` with three AXI-Stream outputs and no direct native EDN endpoints. The crypto and pool paths convert the selected streams back to native EDN requests/responses outside that wrapper.

*Table 191. SEP entropy sink paths*

| Stream / select bit | Sink | Connection after the source mux |
|---|---|---|
| 0 | Key Manager | Direct stream to the KM DRBG sampler. |
| 1 | AES, KMAC, OTBN | `u_axis_edn_crypto_s3c_scan` distributes words to native clients 0 = AES, 1 = KMAC, 2 = OTBN RND, 3 = OTBN URND, using round-robin arbitration. |
| 2 | Fabric entropy pool | `u_axis_edn_pool_s3c_scan` supplies the single native EDN fill port of `u_entropy_fifo` in `sep.sv`. |

`SEP_CPU_CTRL.EXT_TRNG_SRC_SEL` selects each path independently: 0 selects internal DRBG data and 1 selects external data. Its reset value is `0b111`, so all three paths initially select external inputs. The write-once-set `EXT_TRNG_SRC_SEL_LOCK` prevents selection changes until the CPU-control bank is reset; resetting the internal TRNG alone does not unlock it.

The selected source receives the sink's ready signal. An unselected internal endpoint is backpressured and retains queued data; an unselected external input receives ready high and its beats are discarded. Selection is combinational: there is no switch-complete handshake or automatic flush of previously queued words.

*Table 192. SEP random-data stream contract*

| Signal | Direction / width | Meaning |
|---|---|---|
| `tvalid` | Source / 1 | A word is available; transfer occurs with `tready` high. |
| `tready` | Sink / 1 | The sink can accept the word. Hold valid data stable while stalled. |
| `tdata` | Source / 32 | One random-data word. |

| Signal | Direction / width | Meaning |
|---|---|---|
| `tstrb` | Source / 4 | Byte strobes. Internal endpoints drive `0xF`; the crypto and pool adapters accept only full words (`0xF`). Partial strobes stall those adapters. |
| `tuser` | Internal source / 1 | Per-word FIPS provenance from EDN. The external input has no provenance bit; its source mux supplies a constant 1. |

There is no stream address: each physical output selects its sink path. The pool packs received words for its 64-bit read aperture and reports `entropy_pool_low`, `entropy_pool_fill_stall` and `entropy_pool_err`.

See SEP CPU Interrupt Vector Map for source assignments.

### FIPS Provenance

The internal seed adapter attaches the provisional constant `DrbgCsrngSeedFipsProvisional = 1`. CSRNG/EDN output provenance is retained in each endpoint FIFO and forwarded through `tuser` into the native client's `edn_fips`. OTBN RND checks the returned tag and can report `rnd_fips_chk_fail`. The external mux also attaches 1 unconditionally. Neither constant measures source quality, validates an external producer, or establishes FIPS compliance; qualification must justify the source and its operating configuration separately.

### Entropy Access Paths

Random data leaves the TRNG through the entropy pool, which EDN fills from the source selected for stream 2. The entropy source's three readable FIFOs (`FIFO_RDATA`, `BIW_OBS_RDATA` and `NOISE_OBS_RDATA`) expose the DRBG seed words and pre-conditioning entropy, so they are debug interfaces. SEP drives the entropy source's FIFO access enable from the `ENTROPY_DBG` feature-control bit, bit 4 of the DBG_1 group in the disable vector format. The FIFOs are therefore accessible exactly when the per-LC-state feature control profile enables DBG_1 bit 4, and the gate's behaviour is in the Entropy Source chapter (FIFO Access Gate). `ENTROPY_DBG` does not gate the entropy pool, nor the CSRNG application interface, whose enable is `CSRNG.CTRL.SW_APP_ENABLE`.

**NOTE** `ENTROPY_DBG` and its wiring are specified ahead of the hardware. Until the RTL implements them, the three FIFOs are accessible in every lifecycle state, subject only to `FIPS_LOCK`.

### Reset and Recovery Boundaries

`SEP_RESET_CTRL.SW_RESET_N.TRNG_SW_RST_N` controls one internal reset domain: ESRC, CSRNG, EDN and the DRBG endpoint queues. The TRNG sequencer isolates all three host paths to drain and safely reset their converted AXI4-Lite CSR transactions. This protects register transactions; it does not wait for entropy consumers to become idle.

*Table 193. TRNG reset effects outside the internal domain*

| Block | Effect |
|---|---|
| Post-mux crypto and pool adapters | Synchronously clear pending and buffered state while the internal TRNG reset is active, even if external inputs are selected. |
| Fabric entropy pool | Clears its partial word, queued entropy and fill-stall state. Its AXI responder remains accessible; this is not a reset of that responder. |
| AES, KMAC and OTBN | Not reset by the TRNG coordinator. Native requests receive no acknowledgement during adapter clear and can resume after fresh data arrives. Their software resets cancel the corresponding adapter endpoints; OTBN owns RND and URND. |
| Key Manager sampler | Not scrubbed by TRNG reset: there is no post-mux adapter on this path. Keep KM idle before asserting TRNG reset; holding KM in software reset is sufficient. Hardware does not enforce this ordering. |
| External TRNG CSR path | Not included in the internal TRNG reset/isolation boundary. |

An in-progress KM sample can otherwise combine old and new stream data or report a stream error. The reset is therefore not an atomic restart of all entropy consumers, and after release the internal chain produces no data until ESRC, CSRNG and EDN are reinitialized. External crypto/pool traffic also pauses during adapter clear; it can resume when that clear ends, independently of internal generator initialization. The TRNG recovery sequence is in the Programmer's Guide.

See Resetting the TRNG for the coordinator and recovery contract.

#### Scan Protection Boundary

The Class 2 entropy boundary starts at `u_trng_wrapper_s3c_scan`: its sequencer, 128-bit release stage and stream FIFOs retain seed material; the vendor core uses its delivered DFT view. Within `u_sep_crypto.u_sep_trng`, `u_entropy_source_s3c_scan` covers entropy FIFO/SHA-256 state and `u_drbg_s3c_scan` covers CSRNG, EDN and endpoint state. The post-mux `u_axis_edn_crypto_s3c_scan` covers staging outside the DRBG root. Source-selection muxes are combinational, not scan assets.

### DRBG Deterministic Random Bit Generator

This module is NIST 800-90A compliant. The module is based on the OpenTitan design.

The DRBG is seeded by the TRNG entropy stream and generates an arbitrary length bitstream which is used as foundational random data by the Key Generator.

This component uses the AES-CTR-DRBG version of the 800-90A.

The `drbg` top-level module wraps the OpenTitan CSRNG and EDN blocks with bus-protocol adapters. It accepts a raw 32-bit entropy stream from the entropy source, feeds it directly into the CSRNG seed path, and exposes AXI-Stream (and optionally native) EDN endpoints for deterministic random data delivery.

#### Parameters

The DRBG is configured through its seed FIFO depth, its AXI-Stream and native EDN endpoint counts, and the depth of each AXI-Stream endpoint FIFO.

The parameters of `drbg` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### External Interfaces

The DRBG presents several interface types to the rest of the SEP subsystem.

Table 194 describes the entropy ingress, random-data endpoints, and control interfaces.

*Table 194. DRBG external interfaces*

| Interface | Description |
|---|---|
| Entropy ingress | Fire-and-forget: `entropy_stream_data_i[31:0]` and `entropy_stream_vld_i` from the entropy source. There is no backpressure; each valid word feeds the CSRNG seed-packing path directly on the cycle it arrives. |
| EDN AXI-Stream | `edn_axis_o` / `edn_axis_i` (one pair per endpoint). Carries deterministic random data generated by CSRNG through the EDN. This is the primary interface consumed by the Key Manager DRBG sampler. |
| Native EDN | `edn_native_req_i` / `edn_native_rsp_o` (one pair per native endpoint, present when `EDN_NATIVE_ENDPOINT_COUNT > 0`). Raw `edn_pkg` req/rsp handshake for consumers that bypass AXI-Stream buffering, such as the SEP entropy-pool FIFO fill path. |

| Interface | Description |
|---|---|
| CSRNG AXI-Lite (64-bit) | `csrng_axil_req_i` / `csrng_axil_rsp_o`. Control-plane access to the CSRNG register set for instantiation, reseeding, and status monitoring. |
| EDN AXI-Lite (64-bit) | `edn_axil_req_i` / `edn_axil_rsp_o`. Control-plane access to the EDN register set for mode configuration and status. |
| Life-cycle / OTP | `otp_en_csrng_sw_app_read_i` (MuBi8) and `lc_hw_debug_en_i` (lc_tx_t). These signals gate CSRNG software application read access based on device life-cycle state. |
| Alerts | Two alert pairs (CSRNG and EDN) following the OpenTitan alert protocol. |
| Interrupts | CSRNG: command done, entropy request, hardware instance exception, fatal error. EDN: command done, fatal error. |

### Seed Assembly

The `drbg_csrng_seed_adapter` receives one 32-bit entropy word per valid cycle directly from the entropy source (there is no upstream routing or distribution FIFO) and packs the words into full CSRNG-width seeds using an OpenTitan `prim_packer_fifo`. The packed seed is pushed into a shallow FIFO (default depth 1) along with a FIPS compliance flag set to a provisional value (`DrbgCsrngSeedFipsProvisional = 1`). The CSRNG pulls seeds through the standard `entropy_src_hw_if` handshake: `es_req` / `es_ack` / `es_bits` / `es_fips`. Because ingress is fire-and-forget with no backpressure, the producer must supply entropy no faster than the seed adapter can pack it.

### OpenTitan CSRNG and EDN

The CSRNG and EDN instances are standard OpenTitan IP blocks. The CSRNG implements AES-CTR-DRBG per NIST SP 800-90A. It accepts entropy seeds from the hardware interface, processes firmware-issued commands (instantiate, reseed, generate, uninstantiate) through TL-UL, and serves random data to multiple hardware application ports. In this integration, EDN is connected to CSRNG hardware application port 0; any additional hardware application ports are tied off.

The EDN automates generate and reseed command sequences so that downstream consumers receive a continuous stream of random data without firmware intervention. The EDN endpoint interface uses a simple request/acknowledge handshake. The `drbg_edn_axis_adapter` converts this into the AXI-Stream protocol expected by the rest of the SEP fabric, buffering data in per-endpoint FIFOs.

Consult the OpenTitan documentation at CSRNG and EDN for the authoritative core functionality of those IPs.

### Bus Protocol Adaptation

The CSRNG and EDN register blocks use TL-UL, following the OpenTitan convention. Each 64-bit AXI4-Lite control port reaches them through two stages:

1. `drbg_axil64_lane_adapter`: 64-bit AXI4-Lite to 32-bit AXI-Lite, one 32-bit lane per access.
2. `axi_lite_to_tlul`: 32-bit AXI-Lite to TL-UL.

Table 195 lists the accesses the lane adapter forwards. Address bit [2] selects the lane.

*Table 195. DRBG lane adapter forwarded accesses*

| Access | Address bit [2] | Forwarded when | Data |
|---|---|---|---|
| Write | 0 | Address 4-byte aligned, WSTRB `0x0F` | Bits [31:0] |
| Write | 1 | Address 4-byte aligned, WSTRB `0xF0` | Bits [63:32] |
| Read | 0 | Address 4-byte aligned | Bits [31:0]. Bits [63:32] read as zero. |

| Access | Address bit [2] | Forwarded when | Data |
|---|---|---|---|
| Read | 1 | Address 4-byte aligned | Bits [63:32]. Bits [31:0] read as zero. |

- Any other access returns SLVERR and sends no downstream request. A rejected read returns zero data.
- A forwarded access returns the downstream response.
- Write address, write data and read address are independent channels. The adapter holds one downstream 32-bit slot.
- A write half is accepted when that half is not already pending. A read is accepted only when neither write half is pending.
- A read and a complete write in the same idle cycle: the read is forwarded first, and the write is held until the adapter is idle again.

The CSRNG and EDN register maps are reported in the SEP detailed register map.

### Memory Map and Register Reference

#### Memory Map

CSRNG and EDN occupy separate apertures in the SEP memory map. Their register offsets and access rules are given in the generated maps below.

#### Detailed Register Map

#### CSRNG Registers

#### Cryptographically Secure Random Number Generator

Takes entropy bits to produce cryptographically secure random numbers for consumption by hardware blocks and by software

*Table 196. Cryptographically Secure Random Number Generator register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | REGWEN | RW | - |
| 0x14 | CTRL | RW | - |
| 0x18 | CMD_REQ | W | - |
| 0x1C | RESEED_INTERVAL | RW | - |
| 0x20 | RESEED_COUNTER_0 | R | - |
| 0x24 | RESEED_COUNTER_1 | R | - |
| 0x28 | RESEED_COUNTER_2 | R | - |
| 0x2C | SW_CMD_STS | R | - |
| 0x30 | GENBITS_VLD | R | - |
| 0x34 | GENBITS | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x38 | INT_STATE_READ_ENABLE | RW | - |
| 0x3C | INT_STATE_READ_ENABLE_REGWEN | RW | - |
| 0x40 | INT_STATE_NUM | RW | - |
| 0x44 | INT_STATE_VAL | R | - |
| 0x48 | FIPS_FORCE | RW | - |
| 0x4C | HW_EXC_STS | RW | - |
| 0x50 | RECOV_ALERT_STS | RW | - |
| 0x54 | ERR_CODE | R | - |
| 0x58 | ERR_CODE_TEST | RW | - |
| 0x5C | MAIN_SM_STATE | R | - |

### INTR_STATE

*Table 197. csrng.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `CS_FATAL_ERR` | RW1C | 0x0 | Asserted when a FIFO error or a fatal alert occurs. Check the !!ERR_CODE register to get more information. |
| 2 | `CS_HW_INST_EXC` | RW1C | 0x0 | Asserted when a hardware-attached CSRNG instance encounters a command exception |
| 1 | `CS_ENTROPY_REQ` | RW1C | 0x0 | Asserted when a request for entropy has been made. |
| 0 | `CS_CMD_REQ_DONE` | RW1C | 0x0 | Asserted when a command request is completed. |

### INTR_ENABLE

*Table 198. csrng.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `CS_FATAL_ERR` | RW | 0x0 | Enable interrupt when !!INTR_STATE.cs_fatal_err is set. |
| 2 | `CS_HW_INST_EXC` | RW | 0x0 | Enable interrupt when !!INTR_STATE.cs_hw_inst_exc is set. |
| 1 | `CS_ENTROPY_REQ` | RW | 0x0 | Enable interrupt when !!INTR_STATE.cs_entropy_req is set. |
| 0 | `CS_CMD_REQ_DONE` | RW | 0x0 | Enable interrupt when !!INTR_STATE.cs_cmd_req_done is set. |

### INTR_TEST

*Table 199. csrng.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `CS_FATAL_ERR` | W | 0x0 | Write 1 to force !!INTR_STATE.cs_fatal_err to 1. |
| 2 | `CS_HW_INST_EXC` | W | 0x0 | Write 1 to force !!INTR_STATE.cs_hw_inst_exc to 1. |
| 1 | `CS_ENTROPY_REQ` | W | 0x0 | Write 1 to force !!INTR_STATE.cs_entropy_req to 1. |
| 0 | `CS_CMD_REQ_DONE` | W | 0x0 | Write 1 to force !!INTR_STATE.cs_cmd_req_done to 1. |

### ALERT_TEST

*Table 200. csrng.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `FATAL_ALERT` | W | 0x0 | Write 1 to trigger one alert event of this kind. |
| 0 | `RECOV_ALERT` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### REGWEN

*Table 201. csrng.REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `REGWEN` | RW0C | 0x1 | When true, all writeable registers can be modified. When false, they become read-only. |

### CTRL

*Table 202. csrng.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:12 | `FIPS_FORCE_ENABLE` | RW | 0x9 | Setting this field to kMultiBitBool4True enables forcing the FIPS/CC compliance flag to true via the !!FIPS_FORCE register. |
| 11:8 | `READ_INT_STATE` | RW | 0x9 | Setting this field to kMultiBitBool4True will enable reading from the !!INT_STATE_VAL register. Reading the internal state of the enable instances will be enabled only if the otp_en_csrng_sw_app_read input vector is set to the enable encoding. Also, the !!INT_STATE_READ_ENABLE bit of the selected instance needs to be set to true for this to work. |
| 7:4 | `SW_APP_ENABLE` | RW | 0x9 | Setting this field to kMultiBitBool4True will enable reading from the !!GENBITS register. This application interface for software (register based) will be enabled only if the otp_en_csrng_sw_app_read input vector is set to the enable encoding. |
| 3:0 | `ENABLE` | RW | 0x9 | Setting this field to kMultiBitBool4True will enable the CSRNG module. The modules of the entropy complex may only be enabled and disabled in a specific order, see Programmers Guide for details. |

### CMD_REQ

*Table 203. csrng.CMD_REQ fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `CMD_REQ` | W | 0x0 | Writing this request with defined CSRNG commands will initiate all possible CSRNG actions. The application interface must wait for the "ack" to return before issuing new commands. |

### RESEED_INTERVAL

*Table 204. csrng.RESEED_INTERVAL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESEED_INTERVAL | RW | 0xFFFFFFFF | Setting this field will set the number of generate requests that can be made to CSRNG before a reseed request needs to be made. This register supports a maximum of 2^32 requests between reseeds. This register will be compared to a counter, which counts the number of generate commands between reseed or instantiate commands. If the counter reaches the value of this register, the violating command will be acknowledged with a status error. If the violating command was issued by a HW instance, an interrupt will be triggered. |

### RESEED_COUNTER_0

*Table 205. csrng.RESEED_COUNTER_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESEED_COUNTER_0 | R | 0x0 | Reseed Counter indicating the number of completed Generate requests since the last Instantiate or Reseed command. |

### RESEED_COUNTER_1

*Table 206. csrng.RESEED_COUNTER_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESEED_COUNTER_0 | R | 0x0 | For RESEED_COUNTER1 |

### RESEED_COUNTER_2

*Table 207. csrng.RESEED_COUNTER_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESEED_COUNTER_0 | R | 0x0 | For RESEED_COUNTER2 |

### SW_CMD_STS

*Table 208. csrng.SW_CMD_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5:3 | CMD_STS | R | 0x0 | This field represents the status code returned with the application command ack. It is updated each time a command ack is asserted on the internal application interface for software use. To check whether a command was successful, wait for !!INTR_STATE.CS_CMD_REQ_DONE or !!SW_CMD_STS.CMD_ACK to be high and then check the value of this field. |
| 2 | CMD_ACK | R | 0x0 | This one bit field indicates when a SW command has been acknowledged by the CSRNG. It is set to low each time a new command is written to !!CMD_REQ. The field is set to high once a SW command request has been acknowledged by the CSRNG. 0b0: The last SW command has not been acknowledged yet. 0b1: The last SW command has been acknowledged. In case of a generate command the acknowledgement goes high after all of the requested entropy is consumed. |
| 1 | CMD_RDY | R | 0x0 | This bit indicates when the command interface is ready to accept commands. Before starting to write a new command to !!SW_CMD_REQ, this field needs to be polled. 0b0: CSRNG is not ready to accept commands or the last command hasn't been acked yet. 0b1: CSRNG is ready to accept the next command. |
| 0 | Reserved | - | - | Reserved |

### GENBITS_VLD

*Table 209. csrng.GENBITS_VLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `GENBITS_FIPS` | R | - | This bit is set when genbits are FIPS/CC compliant. |
| 0 | `GENBITS_VLD` | R | - | This bit is set when genbits are available on this application interface after a generate command has been issued. |

### GENBITS

*Table 210. csrng.GENBITS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `GENBITS` | R | - | Reading this register will get the generated bits that were requested with the generate request. This register must be read four times for each request made. For example, an application command generate request with a `clen` value of 4 requires this register to be read 16 times to get all of the data out of the FIFO path. Note that for !!GENBITS to be able to deliver random numbers, also !!CTRL.SW_APP_ENABLE needs to be set to `kMultiBitBool4True`. In addition, the otp_en_csrng_sw_app_read input needs to be set to `kMultiBitBool8True`. Otherwise, the register reads as 0. |

### INT_STATE_READ_ENABLE

*Table 211. csrng.INT_STATE_READ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `INT_STATE_READ_ENABLE` | RW | 0x7 | Per-instance internal state read enable. Defines whether the internal state of the corresponding instance is readable via !!INT_STATE_VAL. Note that for !!INT_STATE_VAL to provide read access to the internal state, also !!CTRL.READ_INT_STATE needs to be set to `kMultiBitBool4True`. In addition, the otp_en_csrng_sw_app_read input needs to be set to `kMultiBitBool8True`. |

### INT_STATE_READ_ENABLE_REGWEN

*Table 212. csrng.INT_STATE_READ_ENABLE_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `INT_STATE_READ_ENABLE_REGWEN` | RW0C | 0x1 | INT_STATE_READ_ENABLE register configuration enable bit. If this is cleared to 0, the INT_STATE_READ_ENABLE register cannot be written anymore. |

### INT_STATE_NUM

*Table 213. csrng.INT_STATE_NUM fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3:0 | `INT_STATE_NUM` | RW | 0x0 | Setting this field will set the number for which internal state can be selected for a read access. Up to 16 internal state values can be chosen from this register. The actual number of valid internal state fields is set by parameter NumHwApps plus 1 software app. For those selections that point to reserved locations (greater than NumHwApps plus 1), the returned value will be zero. Writing this register will also reset the internal read pointer for the !!INT_STATE_VAL register. Note: This register should be read back after being written to ensure that the !!INT_STATE_VAL read back is accurate. |

### INT_STATE_VAL

*Table 214. csrng.INT_STATE_VAL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `INT_STATE_VAL` | R | - | Reading this register will dump out the contents of the selected internal state field. Since the internal state field is 448 bits wide, it will require 14 reads from this register to gather the entire field. Once 14 reads have been done, the internal read pointer (selects 32 bits of the 448 bit field) will reset to zero. The !!INT_STATE_NUM can be re-written at this time (internal read pointer is also reset), and then another internal state field can be read. Note that for !!INT_STATE_VAL to provide read access to the internal state, also !!CTRL.READ_INT_STATE needs to be set to `kMultiBitBool4True`. In addition, the otp_en_csrng_sw_app_read input needs to be set to `kMultiBitBool8True`. Also, the !!INT_STATE_READ_ENABLE bit of the selected instance needs to be set to true for this to work. Otherwise, the register reads as 0. |

### FIPS_FORCE

*Table 215. csrng.FIPS_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `FIPS_FORCE` | RW | 0x0 | Force the FIPS/CC compliance flag of individual instances to true. This allows CSRNG to set the output FIPS/CC compliance flag to true despite running in fully deterministic mode (flag0 being true). This can be useful e.g. for known-answer testing through entropy consumers accepting FIPS/CC compliant entropy only, or when firmware is used to derive FIPS/CC compliant entropy seeds. After setting a particular bit to 1, the FIPS/CC compliance flag of the corresponding instance will be forced to true upon the next Instantiate or Reseed command.<br><br>Note that for this to work, !!CTRL.FIPS_FORCE_ENABLE needs to be set to kMultiBitBool4True. |

### HW_EXC_STS

*Table 216. csrng.HW_EXC_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `HW_EXC_STS` | RW0C | 0x0 | Reading this register indicates whether one of the CSRNG HW instances has encountered an exception. Each bit corresponds to a particular hardware instance, with bit 0 corresponding to instance HW0, bit 1 corresponding to instance HW1, and so forth. (To monitor the status of requests made to the SW instance, check the !!SW_CMD_STS register). Writing a zero to this register resets the status bits. |

### RECOV_ALERT_STS

*Table 217. csrng.RECOV_ALERT_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15 | `CMD_STAGE_RESEED_CNT_ALERT` | RW0C | 0x0 | This bit is set when the maximum number of generate requests between reseeds is exceeded. The invalid generate command is ignored and CSRNG continues to operate. Writing a zero resets this status bit. |
| 14 | `CMD_STAGE_INVALID_CMD_SEQ_ALERT` | RW0C | 0x0 | This bit is set when an out of order command is received by the main state machine. This happens when an instantiate command is sent for a state that was already instantiated or when any command other than instantiate is sent for a state that wasn't instantiated yet. The invalid command is ignored and CSRNG continues to operate. Writing a zero resets this status bit. |
| 13 | `CMD_STAGE_INVALID_ACMD_ALERT` | RW0C | 0x0 | This bit is set when an unsupported/illegal CSRNG command is received by the main state machine. The invalid command is ignored and CSRNG continues to operate. Writing a zero resets this status bit. |
| 12 | `CS_BUS_CMP_ALERT` | RW0C | 0x0 | This bit is set when the software application port genbits bus value is equal to the prior valid value on the bus, indicating a possible attack. Writing a zero resets this status bit. |
| 11:5 | `Reserved` | - | - | Reserved |
| 4 | `ACMD_FLAG0_FIELD_ALERT` | RW0C | 0x0 | This bit is set when the FLAG0 field in the Application Command is set to a value other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 3 | `FIPS_FORCE_ENABLE_FIELD_ALERT` | RW0C | 0x0 | This bit is set when the FIPS_FORCE_ENABLE field in the !!CTRL register is set to a value other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 2 | `READ_INT_STATE_FIELD_ALERT` | RW0C | 0x0 | This bit is set when the READ_INT_STATE field in the !!CTRL register is set to a value other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 1 | `SW_APP_ENABLE_FIELD_ALERT` | RW0C | 0x0 | This bit is set when the SW_APP_ENABLE field in the !!CTRL register is set to a value other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 0 | `ENABLE_FIELD_ALERT` | RW0C | 0x0 | This bit is set when the ENABLE field in the !!CTRL register is set to a value other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |

### ERR_CODE

*Table 218. csrng.ERR_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30 | `FIFO_STATE_ERR` | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 15 of this this register) are asserted as a result of an error pulse generated from any FIFO where both the empty and full status bits are set. This bit will stay set until the next reset. |
| 29 | `FIFO_READ_ERR` | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 15 of this this register) are asserted as a result of an error pulse generated from any empty FIFO that has received a read pulse. This bit will stay set until the next reset. |
| 28 | `FIFO_WRITE_ERR` | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 15 of this this register) are asserted as a result of an error pulse generated from any full FIFO that has been received a write pulse. This bit will stay set until the next reset. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 27 | `Reserved` | - | - | Reserved |
| 26 | `CTR_ERR` | R | 0x0 | This bit will be set to one when a mismatch in any of the hardened counters has been detected. This error will signal a fatal alert, and also an interrupt if enabled. This bit will stay set until the next reset. |
| 25 | `AES_CIPHER_SM_ERR` | R | 0x0 | This bit will be set to one when an AES fatal error has been detected. This error will signal a fatal alert, and also an interrupt if enabled. This bit will stay set until the next reset. |
| 24:23 | `Reserved` | - | - | Reserved |
| 22 | `CTR_DRBG_SM_ERR` | R | 0x0 | This bit will be set to one when an illegal state has been detected for the ctr_drbg state machine. This error will signal a fatal alert, and also an interrupt if enabled. This bit will stay set until the next reset. |
| 21 | `MAIN_SM_ERR` | R | 0x0 | This bit will be set to one when an illegal state has been detected for the main state machine. This error will signal a fatal alert, and also an interrupt if enabled. This bit will stay set until the next reset. |
| 20 | `CMD_STAGE_SM_ERR` | R | 0x0 | This bit will be set to one when an illegal state has been detected for the command stage state machine. This error will signal a fatal alert, and also an interrupt if enabled. This bit will stay set until the next reset. |
| 19:2 | `Reserved` | - | - | Reserved |
| 1 | `SFIFO_GENBITS_ERR` | R | 0x0 | This bit will be set to one when an error has been detected for the command stage genbits FIFO. The type of error is reflected in the type status bits (bits 28 through 30 of this register). This bit will stay set until the next reset. |
| 0 | `SFIFO_CMD_ERR` | R | 0x0 | This bit will be set to one when an error has been detected for the command stage command FIFO. The type of error is reflected in the type status bits (bits 28 through 30 of this register). This bit will stay set until the next reset. |

#### ERR_CODE_TEST

*Table 219. csrng.ERR_CODE_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `ERR_CODE_TEST` | RW | 0x0 | Setting this field will set the bit number for which an error will be forced in the hardware. This bit number is that same one found in the !!ERR_CODE register. The action of writing this register will force an error pulse. The sole purpose of this register is to test that any error properly propagates to either an interrupt or an alert. |

#### MAIN_SM_STATE

*Table 220. csrng.MAIN_SM_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `MAIN_SM_STATE` | R | 0x37 | This is the state of the CSRNG main state machine. See the RTL file `csrng_main_sm` for the meaning of the values. |

### EDN Registers

### Entropy Distribution Network

Distributes random numbers produced by CSRNG to hardware blocks

*Table 221. Entropy Distribution Network register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | REGWEN | RW | - |
| 0x14 | CTRL | RW | - |
| 0x18 | BOOT_INS_CMD | RW | - |
| 0x1C | BOOT_GEN_CMD | RW | - |
| 0x20 | SW_CMD_REQ | W | - |
| 0x24 | SW_CMD_STS | R | - |
| 0x28 | HW_CMD_STS | R | - |
| 0x2C | RESEED_CMD | W | - |
| 0x30 | GENERATE_CMD | W | - |
| 0x34 | MAX_NUM_REQS_BETWEEN_RESEEDS | RW | - |
| 0x38 | RECOV_ALERT_STS | RW | - |
| 0x3C | ERR_CODE | R | - |
| 0x40 | ERR_CODE_TEST | RW | - |
| 0x44 | MAIN_SM_STATE | R | - |

### INTR_STATE

*Table 222. edn.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `edn_fatal_err` | RW1C | 0x0 | Asserted when a FIFO error occurs. |
| 0 | `edn_cmd_req_done` | RW1C | 0x0 | Asserted when a software CSRNG request has completed. |

### INTR_ENABLE

*Table 223. edn.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `edn_fatal_err` | RW | 0x0 | Enable interrupt when !!INTR_STATE.edn_fatal_err is set. |
| 0 | `edn_cmd_req_done` | RW | 0x0 | Enable interrupt when !!INTR_STATE.edn_cmd_req_done is set. |

### INTR_TEST

*Table 224. edn.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `edn_fatal_err` | W | 0x0 | Write 1 to force !!INTR_STATE.edn_fatal_err to 1. |
| 0 | `edn_cmd_req_done` | W | 0x0 | Write 1 to force !!INTR_STATE.edn_cmd_req_done to 1. |

### ALERT_TEST

*Table 225. edn.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `fatal_alert` | W | 0x0 | Write 1 to trigger one alert event of this kind. |
| 0 | `recov_alert` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### REGWEN

*Table 226. edn.REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `REGWEN` | RW0C | 0x1 | When true, the CTRL can be written by software. When false, this field read-only. Defaults true, write zero to clear. Note that this needs to be cleared after initial configuration at boot in order to lock in the listed register settings. |

### CTRL

*Table 227. edn.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:12 | `CMD_FIFO_RST` | RW | 0x9 | Setting this field to kMultiBitBool4True clears the two command FIFOs: the RESEED_CMD FIFO and the GENERATE_CMD FIFO. This field must be set to the reset state by software before any further commands can be issued to these FIFOs. |
| 11:8 | `AUTO_REQ_MODE` | RW | 0x9 | Setting this field to kMultiBitBool4True enables auto request mode. In this mode, EDN automatically sends `generate` and `reseed` command requests to the CSRNG application interface. The purpose of this mode is to continuously deliver entropy to endpoints without firmware intervention.<br><br>For this to work, firmware has to 1) configure the !!GENERATE_CMD, !!RESEED_CMD, and !!MAX_NUM_REQS_BETWEEN_RESEEDS registers, and 2) to issue the first `instantiate` command via the !!SW_CMD_REQ register. Once this command has been acknowledged by CSRNG, the first `generate` command is sent out automatically, and a `reseed` command is sent after every MAX_NUM_REQS_BETWEEN_RESEEDS number of `generate` commands.<br><br>Note that the BOOT_REQ_MODE field takes precedence over this field: If both fields are set, EDN enters boot-time request mode. If none of the fields are set, EDN enters Software Port Mode upon enabling. |
| 7:4 | `BOOT_REQ_MODE` | RW | 0x9 | Setting this field to kMultiBitBool4True enables the boot-time request mode. In this mode, EDN automatically sends a boot-time request to the CSRNG application interface. The purpose of this mode is to request entropy as fast as possible after reset, and during chip boot time.<br><br>Note that this takes precedence over the AUTO_REQ_MODE field: If both fields are set, EDN enters boot-time request mode. If none of the fields are set, EDN enters Software Port Mode upon enabling. |
| 3:0 | `EDN_ENABLE` | RW | 0x9 | Setting this field to kMultiBitBool4True enables the EDN module. The modules of the entropy complex may only be enabled and disabled in a specific order, see Programmers Guide for details. |

### BOOT_INS_CMD

*Table 228. edn.BOOT_INS_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `BOOT_INS_CMD` | RW | 0x901 | This field is used as the value for the `instantiate` command at boot time.<br><br>See [Command Header](../../csrng/doc/theory_of_operation.md#command-header) for the meaning of the individual bits. Note that the hardware only supports a value of 0 for the `clen` field. If `clen` has a different value, EDN will hang. Fixing this requires disabling and restarting both EDN and CSRNG. |

### BOOT_GEN_CMD

*Table 229. edn.BOOT_GEN_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `BOOT_GEN_CMD` | RW | 0xFFF003 | This field is used as the value for the `generate` command at boot time.<br><br>See [Command Header](../../csrng/doc/theory_of_operation.md#command-header) for the meaning of the individual bits. Note that the hardware only supports a value of 0 for the `clen` field. If `clen` has a different value, EDN will hang. Fixing this requires disabling and restarting both EDN and CSRNG. |

### SW_CMD_REQ

*Table 230. edn.SW_CMD_REQ fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `SW_CMD_REQ` | W | - | Any CSRNG action can be initiated by writing a CSRNG command to this register. Before any write operation to this register, firmware must read !!SW_CMD_STS to check whether EDN is ready to receive a new command or the next word of a previously started command.<br><br>While !!CTRL.AUTO_REQ_MODE is set, only the first instantiate command has any effect. After that command has been processed, writes to this register will have no effect on operation, until !!CTRL.AUTO_REQ_MODE is de-asserted and the state machine of EDN enters the `SwPortMode` state.<br><br>Refer to the [CSRNG documentation](../../csrng/doc/theory_of_operation.md#general-command-format) for details on the command format. |

### SW_CMD_STS

*Table 231. edn.SW_CMD_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:3 | `CMD_STS` | R | 0x0 | This field represents the status code returned with the CSRNG application command ack. It is updated each time a SW command is acknowledged by CSRNG. To check whether a command was successful, wait for !!INTR_STATE.EDN_CMD_REQ_DONE or !!SW_CMD_STS.CMD_ACK to be high and then check the value of this field. A description of the command status types can be found [here](../../csrng/doc/registers.md#sw_cmd_sts—cmd_sts). |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | `CMD_ACK` | R | 0x0 | This one bit field indicates when a SW command has been acknowledged by the CSRNG. It is set to low each time a new command is written to !!SW_CMD_REQ. The field is set to high once a SW command request has been acknowledged by the CSRNG. 0b0: The last SW command has not been acknowledged yet. 0b1: The last SW command has been acknowledged. |
| 1 | `CMD_RDY` | R | 0x0 | This bit indicates when the EDN is ready to accept the next command. Before starting to write a new command to !!SW_CMD_REQ, this field needs to be polled. 0b0: The EDN is not ready to accept commands or the last command hasn't been acked yet. 0b1: The EDN is ready to accept the next command. |
| 0 | `CMD_REG_RDY` | R | 0x0 | This bit indicates when !!SW_CMD_REQ is ready to accept the next word. This bit has to be polled before each word of a command is written to !!SW_CMD_REQ. 0b0: The EDN is not ready to accept the next word yet. 0b1: The EDN is ready to accept the next word. |

### HW_CMD_STS

*Table 232. edn.HW_CMD_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:7 | `CMD_STS` | R | 0x0 | This field represents the status code returned with the CSRNG application command ack. It is updated each time a HW command is acknowledged by CSRNG. A description of the command status types can be found [here](../../csrng/doc/registers.md#sw_cmd_sts—cmd_sts). |
| 6 | `CMD_ACK` | R | 0x0 | This one bit field indicates when a HW command has been acknowledged by the CSRNG. It is set to low each time a new command is sent to the CSRNG. The field is set to high once a HW command request has been acknowledged by the CSRNG. 0b0: The last HW command has not been acknowledged yet. 0b1: The last HW command has been acknowledged. |
| 5:2 | `CMD_TYPE` | R | 0x0 | This field contains the application command type of the hardware controlled command issued last. The application command selects one of five operations to perform. A description of the application command types can be found [here](../../csrng/doc/theory_of_operation.md#command-description). |
| 1 | `AUTO_MODE` | R | 0x0 | This one bit field indicates whether the EDN is in the hardware controlled part of auto mode. The instantiate command is issued via SW interface and is thus not part of the hardware controlled part of auto mode. 0b0: The EDN is not in the hardware controlled part of auto mode. 0b1: The EDN is in the hardware controlled part of auto mode. |
| 0 | `BOOT_MODE` | R | 0x0 | This one bit field indicates whether the EDN is in the hardware controlled boot mode. 0b0: The EDN is not in boot mode. 0b1: The EDN is in boot mode. |

### RESEED_CMD

*Table 233. edn.RESEED_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESEED_CMD | W | - | Writing this register will fill a FIFO with up to 13 command words (32b words). When running in auto request mode, this FIFO is used to automatically send out a reseed command to the CSRNG application interface after every MAX_NUM_REQS_BETWEEN_RESEEDS number of generate commands.<br><br>See [General Command Format](../../csrng/doc/theory_of_operation.md#general-command-format) for details about the command format.<br><br>Note that the number of additional data words provided must match the value of the clen field of the first word. Otherwise, undefined behavior may result. If more than 13 entries are written to the FIFO, they are ignored and EDN signals an edn_fatal_err interrupt as well as a fatal alert. |

### GENERATE_CMD

*Table 234. edn.GENERATE_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | GENERATE_CMD | W | - | Writing this register will fill a FIFO with up to 13 command words (32b words). When running auto request mode, this FIFO is used to automatically send out generate commands to the CSRNG application interface.<br><br>See [General Command Format](../../csrng/doc/theory_of_operation.md#general-command-format) for details about the command format.<br><br>Note that the number of additional data words provided must match the value of the clen field of the first word. Otherwise, undefined behavior may result. If more than 13 entries are written to the FIFO, they are ignored and EDN signals an edn_fatal_err interrupt as well as a fatal alert. |

### MAX_NUM_REQS_BETWEEN_RESEEDS

*Table 235. edn.MAX_NUM_REQS_BETWEEN_RESEEDS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | MAX_NUM_REQS_BETWEEN_RESEEDS | RW | 0x0 | Setting this field will set the number of generate command requests that are made to CSRNG before a reseed request is made. This value only has meaning when running in auto request mode. This register supports a maximum of 2^32 generate requests between reseeds. This register will be used by a counter that counts down, triggering an automatic reseed request when it reaches zero.<br><br>Note that this value must be chosen smaller than or equal to the value configured in the [RESEED_INTERVAL register of CSRNG](../../csrng/doc/registers.md#reseed-interval). |

### RECOV_ALERT_STS

*Table 236. edn.RECOV_ALERT_STS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:14 | Reserved | - | - | Reserved |
| 13 | CSRNG_ACK_ERR | RW0C | 0x0 | This bit is set when the CSRNG returns an acknowledgement where the status signal is non-zero. Writing a zero resets this status bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 12 | EDN_BUS_CMP_ALERT | RW0C | 0x0 | This bit is set when the interal entropy bus value is equal to the prior valid value on the bus, indicating a possible attack. Writing a zero resets this status bit. |
| 11:4 | Reserved | - | - | Reserved |
| 3 | CMD_FIFO_RST_FIELD_ALERT | RW0C | 0x0 | This bit is set when the CMD_FIFO_RST field is set to an illegal value, something other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 2 | AUTO_REQ_MODE_FIELD_ALERT | RW0C | 0x0 | This bit is set when the !!CTRL.AUTO_REQ_MODE field is set to an illegal value, something other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 1 | BOOT_REQ_MODE_FIELD_ALERT | RW0C | 0x0 | This bit is set when the BOOT_REQ_MODE field is set to an illegal value, something other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |
| 0 | EDN_ENABLE_FIELD_ALERT | RW0C | 0x0 | This bit is set when the EDN_ENABLE field is set to an illegal value, something other than kMultiBitBool4True or kMultiBitBool4False. Writing a zero resets this status bit. |

### ERR_CODE

*Table 237. edn.ERR_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | Reserved | - | - | Reserved |
| 30 | FIFO_STATE_ERR | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 1 of this register) are asserted as a result of an error pulse generated from any FIFO where both the empty and full status bits are set or in case of error conditions inside the hardened counters. This bit will stay set until the next reset. |
| 29 | FIFO_READ_ERR | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 1 of this register) are asserted as a result of an error pulse generated from any empty FIFO that has received a read pulse. This bit will stay set until the next reset. |
| 28 | FIFO_WRITE_ERR | R | 0x0 | This bit will be set to one when any of the source bits (bits 0 through 1 of this register) are asserted as a result of an error pulse generated from any full FIFO that has received a write pulse. This bit will stay set until the next reset. |
| 27:23 | Reserved | - | - | Reserved |
| 22 | EDN_CNTR_ERR | R | 0x0 | This bit will be set to one when a hardened counter has detected an error condition. This error will signal a fatal alert. This bit will stay set until the next reset. |
| 21 | EDN_MAIN_SM_ERR | R | 0x0 | This bit will be set to one when an illegal state has been detected for the EDN main stage state machine. This error will signal a fatal alert. This bit will stay set until the next reset. |
| 20 | EDN_ACK_SM_ERR | R | 0x0 | This bit will be set to one when an illegal state has been detected for the EDN ack stage state machine. This error will signal a fatal alert. This bit will stay set until the next reset. |
| 19:2 | Reserved | - | - | Reserved |
| 1 | SFIFO_GENCMD_ERR | R | 0x0 | This bit will be set to one when an error has been detected for the generate command FIFO. The type of error is reflected in the type status bits (bits 28 through 30 of this register). When this bit is set, a fatal error condition will result. This bit will stay set until the next reset. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `SFIFO_RESCMD_ERR` | R | 0x0 | This bit will be set to one when an error has been detected for the reseed command FIFO. The type of error is reflected in the type status bits (bits 28 through 30 of this register). When this bit is set, a fatal error condition will result. |

#### ERR_CODE_TEST

*Table 238. edn.ERR_CODE_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `ERR_CODE_TEST` | RW | 0x0 | Setting this field will set the bit number for which an error will be forced in the hardware. This bit number is that same one found in the !!ERR_CODE register. The action of writing this register will force an error pulse. The sole purpose of this register is to test that any error properly propagates to either an interrupt or an alert. |

#### MAIN_SM_STATE

*Table 239. edn.MAIN_SM_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8:0 | `MAIN_SM_STATE` | R | 0xC1 | This is the state of the EDN main state machine. See the RTL file `edn_main_sm` for the meaning of the values. |

### Entropy Source

The entropy source component is a random number generator which is based on multiple asynchronous ring oscillators sampled in such a way to generate random bits from the jitter noise and metastable latching behavior. It forms the basis of a True Random Number Generator (TRNG).

The generated stream is ``raw'' noise in the sense that there may still be some statistical biases present which need to be filtered out by an optional entropy conditioner post processing unit which would consume the output noise bits. NIST specifies a set of acceptable conditioners. Common conditioners are based on standard hash functions such as SHA256 and ASCON as well as standard keyed hash functions such as AES-CMAC and HMAC.

The source includes a configurable SHA-256 whitener. Software can bypass whitening to observe the unconditioned stream; health tests also monitor the individual noise generators and the combined stream.

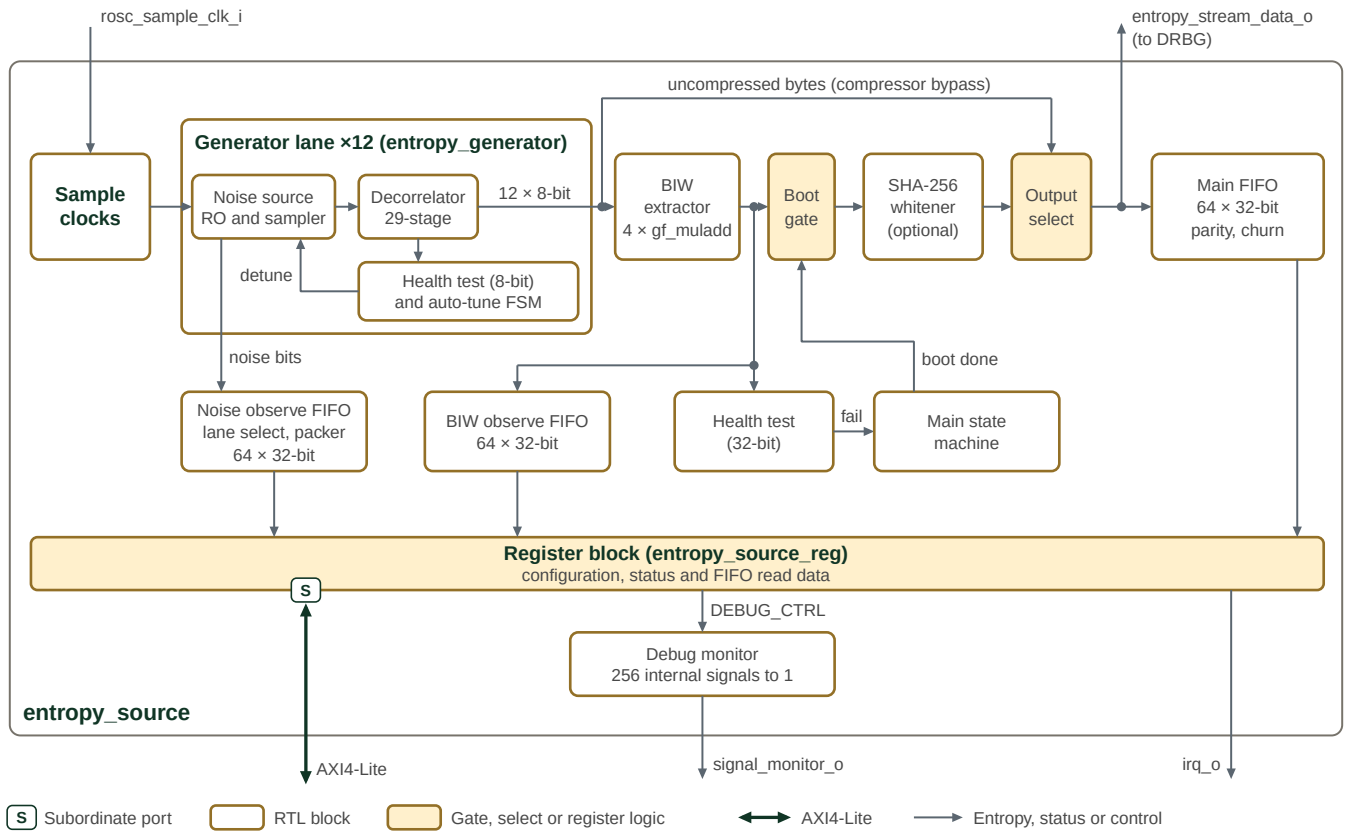


*Figure 34. Entropy source microarchitecture*

Figure 34 shows the entropy source and its conditioning pipeline. The client controls the component via the AXI4-Lite bus interface which provides access to a set of memory mapped configuration and status registers (CSRs) described later in this document. The noise generators feed the decorrelation, mixing, health-test, and optional whitening stages. Each word written into the main FIFO also leaves the block as `entropy_stream_data_o`, qualified by `entropy_stream_vld_o`, towards the DRBG; the FIFO itself is drained through the `FIFO_RDATA` register, behind the access gate in FIFO Access Gate.

The main datapath in the figure runs left to right along the top row. The generator lane stands for all twelve lanes, and Figure 35 shows its noise source and sample clock in detail. The boot gate passes entropy only after the main state machine has seen a passing boot health-test window; it qualifies the whitener input, the uncompressed-byte path and the BIW observe FIFO, while the 32-bit health test and the noise observe FIFO see the stream ungated. The output select takes the whitener output, or, when `CTRL.BYPASS_ENTROPY_COMPRESSOR` is set, the twelve uncompressed decorrelator bytes as three 32-bit words (packing order), downsampled by `CTRL.DOWNSAMPLE_RATE`. The 32-bit health test is clock-gated off in that mode. The register block configures every stage; its connections to individual stages are not drawn. The debug monitor routes one of 256 internal signals to `signal_monitor_o`.

The health-test modules are OpenTitan's `entropy_src` repetition count, adaptive proportion and Markov tests, and the main state machine is OpenTitan's `entropy_src_main_sm`. Their internal structure is described in the OpenTitan entropy source documentation. The Programmer's Guide gives the programming guidance for this block.

The entropy source is designed according to guidelines set forth in the NIST SP800-90B specification and is planned to be ready for FIPS 140-2 and 140-3 certification. It will also be compliant with AIS31 statistical requirements specified by the BSI agency.

### Noise Generation

The entropy source uses 12 independent ring oscillators as its noise source. Each ring oscillator is constructed from the

shared OCAH primitives (prim_clock_nand2, prim_buf, prim_stdmux2) to preserve the asynchronous feedback-loop structure through technology mapping. Each oscillator has two configurable lengths: a TOTAL_LENGTH and a TAPPED_LENGTH, where a multiplexer selects between the short and long feedback paths. The entropy_rosc_tune_fsm toggles between these two lengths on a rising edge of any health test failure, providing a rudimentary frequency detuning mechanism.

The oscillator output is sampled by a metastable D flip-flop (prim_flop) clocked by a sampling clock. The sampled bit is then synchronized to the system clock domain through a two-stage synchronizer. Each lane can be clocked either by an external sampling clock (rosc_sample_clk_i) or by a shared internal ring oscillator (total length 109, tapped length 53). A per-lane ripple divider provides division ratios of 1, 2, 4, 8, 16, and 32, selected through the CSR. Both selects are clock mux cells. Software changes a lane's source or division only while that generator is disabled. A division value outside 0-5 selects divide-by-4.

Figure 35 shows the ring oscillators, sampling paths, and clock selection used to generate noise.

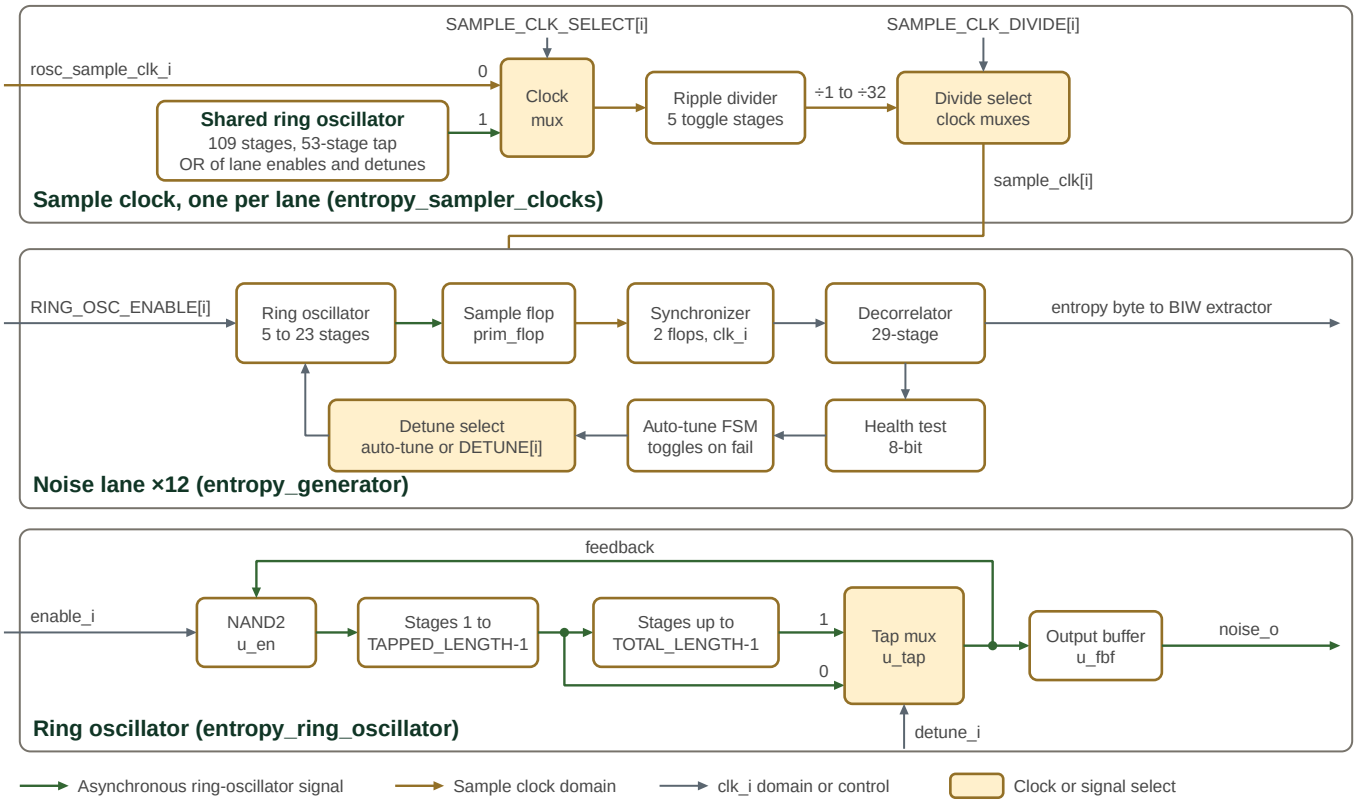


*Figure 35. Entropy source ring-oscillator complex*

Figure 35 has three bands. The top band builds one sample clock per lane: RING_OSC_CTRL.SAMPLE_CLK_SELECT bit *i* chooses rosc_sample_clk_i (0) or the shared ring oscillator (1), and GENERATOR_i_SAMPLE_CLK_CONFIG.SAMPLE_CLK_DIVIDE picks one ripple-divider tap. The middle band is one of the twelve noise lanes; when CTRL.AUTOTUNE_ENABLE is set the lane detune follows the auto-tune FSM, otherwise RING_OSC_TUNE.DETUNE bit *i*. The bottom band is the ring oscillator used both in each lane and, with 109 and 53 stages, as the shared sampling oscillator. Green lines carry asynchronous oscillator signals, brown lines are in the sample-clock domain and grey lines are in the clk_i domain or are controls. Table 240 lists the lane ring lengths.

*Table 240. Noise ring-oscillator lengths per lane*

| Lane | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOTAL_LENGTH (detune 1) | 5 | 7 | 11 | 13 | 17 | 19 | 6 | 8 | 12 | 15 | 18 | 23 |
| TAPPED_LENGTH (detune 0) | 3 | 5 | 9 | 10 | 14 | 16 | 4 | 7 | 11 | 13 | 15 | 21 |

### Decorrelation and BIW Mixing

Each sampled noise bit enters a serial decorrelator (`entropy_decorrelator`), which shifts bits through a 29-stage shift register with XOR feedback from the last stage, as Figure 36 shows. A downsample counter reloaded from `DECORRELATOR_CTRL.SAMPLE_CLK_DIV` latches the top 8 bits of the shift register as an entropy byte every `SAMPLE_CLK_DIV` + 1 `clk_i` cycles, 64 cycles at the reset value of 63, and pulses the byte-valid strobe for one cycle. The byte is ANDed with `DECORRELATOR_MASK.ENTROPY_BYTE_MASK`, so a cleared mask bit reads as zero; the mask and the divider are shared by all 12 lanes. Setting a lane's bit in `DECORRELATOR_CTRL.BYPASS` forces the feedback to zero, so raw noise bits shift through the chain unmixed; the byte is still taken from the top 8 stages, downsampled and masked.

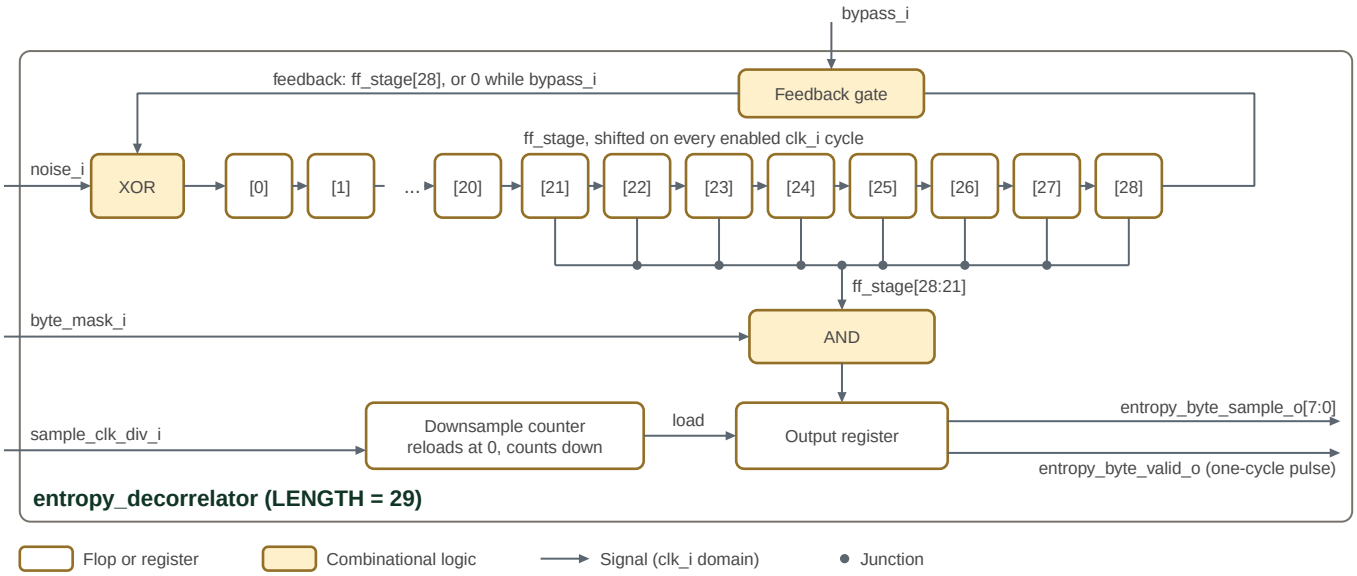


*Figure 36. Entropy source decorrelator*

Figure 36 shows one lane's decorrelator. The chain shifts on every `clk_i` cycle while `enable_i` is high, and the eight taps from `ff_stage[21]` to `ff_stage[28]` join into one byte-wide bundle. The output register loads when the counter reaches zero. While `enable_i` is low, the chain holds its contents and the counter reloads from `sample_clk_div_i`.

The 12 decorrelated byte streams are combined by the Barak-Impagliazzo-Wigderson (BIW) stage, which uses four `gf_muladd` instances performing y = a*b + c in GF(2^8) with the AES field polynomial (0x1B). The 12 input bytes are grouped into three sets of four and reduced to a single 32-bit word. This mixing step improves the statistical independence of the output without introducing significant latency.

With `CTRL.BYPASS_ENTROPY_COMPRESSOR` set, the BIW stage is skipped and each selected decorrelator output reaches the output select as three 32-bit words, written to the FIFO in the order word 0, word 1, word 2. Word $n$ carries the bytes of generators $4n$ to $4n+3$, with generator $4n+k$ in bits $[8k+7:8k]$: word 0 is {generator 3, generator 2, generator 1, generator 0}, with generator 0 in bits [7:0], word 1 carries generators 7 to 4 and word 2 generators 11 to 8. All three words come from the same decorrelator output.

### Health Testing

Health tests operate at two levels. Each of the 12 `entropy_generator` instances runs an 8-bit `entropy_health_test` on its individual decorrelated output. At the top level, a second 32-bit `entropy_health_test` instance monitors the post-BIW (or post-bypass) stream.

The health test wrapper instantiates three OpenTitan-derived test modules. The repetition count test (`entropy_src_repcnt_ht`) tracks consecutive identical values per bit lane and flags when a threshold is exceeded. The adaptive proportion test (`entropy_src_adaptp_ht`) accumulates per-bit ones counts within a configurable window and

compares the maximum and minimum lane counts against high and low thresholds. The Markov test (`entropy_src_markov_ht`) counts bit-pair transitions and compares to high and low thresholds at window boundaries. Window synchronization is driven by a `prim_count` counter in the `entropy_generator_complex`.

Health test thresholds, enable bits, and window sizes are all programmable through the CSR interface. Test status is reported as an 8-bit vector where bit 0 indicates a repetition count failure, bit 3 an adaptive proportion failure, and bits 4-5 Markov high/low failures.

#### Health Test Watermark

`HT_WATERMARK` is one shared register whose source and maximum/minimum direction are selected by `HT_WATERMARK_NUM`; there is no independent storage per selector. A selector write changes which counter updates the shared register but does not clear it, so changing only the selector carries the value recorded by the previous mode.

The shared watermark is armed by a `CTRL.MODULE_ENABLE` 0-to-1 transition. High modes (`REPCNT_HI`, `APT_HI`, and `MARKOV_HI`) arm to `0x0000`; low modes (`APT_LO` and `MARKOV_LO`) arm to `0xFFFF`. A hardware reset restores the selector and watermark together, so a deterministic baseline requires `HT_WATERMARK_NUM` to be written before the `MODULE_ENABLE` toggle. With the reset-default `MODULE_ENABLE=1`, the power-on enable pulse performs the first mode-aware arm automatically.

#### Persistent Failure and AlertHang

The health-test subsystem includes a persistent-failure path that halts entropy output when failures persist beyond a programmed threshold. A saturating counter (`ALERT_SUMMARY_FAIL_COUNTS.ANY_FAIL_COUNT`) records one count per health-test window in which any test failed; a mid-window failure of the continuous repetition-count test is captured and counted once for that window. When the counter reaches `ALERT_THRESHOLD.THRESHOLD` (default 4, i.e. four failing windows, locked by `FIPS_LOCK`), `alert_thresh_fail` is asserted and the main FSM transitions through `AlertState` into `AlertHang`, halting entropy output. `AlertHang` persists until the FSM is reset by `rst_ni` or software clears `CTRL.MODULE_ENABLE`. The event sets `MAIN_SM_STATUS.ALERT` and raises `INTR_STATUS.PERSISTENT_FAILURE`. The sticky `MAIN_SM_STATUS.ALERT` and `MAIN_SM_STATUS.ERR` status bits are cleared by writing 1 (W1C) or by the full entropy-source reset.

Setting `ALERT_THRESHOLD.THRESHOLD` to zero disables the comparator, leaving the counter accumulating without ever asserting `alert_thresh_fail`.

#### SHA-256 Whitening

When the SHA-256 whitener is enabled via the CSR, the 32-bit BIW output stream feeds into the `entropy_sha256_whitener` module. This module accumulates 16 consecutive 32-bit words (one SHA-256 block of 512 bits), computes the SHA-256 digest using the OpenTitan `prim_sha2_32` primitive, and emits the 8-word (256-bit) digest. The compression ratio is 2:1 (16 input words produce 8 output words). When the whitener is disabled, the BIW output passes through to the FIFO without modification.

#### Output FIFO

The conditioned entropy stream is stored in a 64-entry FIFO (`entropy_fifo`). Each entry is 36 bits wide: 32 data bits plus 4 byte-wise odd parity bits. The FIFO includes pointer integrity checking (normal and inverted pointer pairs with mismatch detection) and parity verification on reads. An optional churn mode XORs the incoming data with a word read from the diametrically opposite FIFO position, improving entropy distribution. Overflow, underflow, parity error, and pointer error conditions are all reported to the CSR block and can trigger interrupts.

The `entropy_stream_data_o` and `entropy_stream_vld_o` top-level ports carry each word as it is written into the FIFO, not the FIFO read data, and there is no backpressure. In SEP they connect to the DRBG seed path, which distributes the generated random data to the entropy consumers.

Because the FIFO holds the same words that seed the DRBG, `FIFO_CTRL.ENABLE` is covered by `FIPS_LOCK`. The field gates only the FIFO write, so the DRBG seed stream continues whatever its value. It resets to one, keeping the `FIFO_RDATA` read path live out of reset; the TRNG bring-up sequence in the Programmer's Guide closes it before `FIPS_LOCK`. Reads remain available while writes are disabled, so words already in the FIFO stay readable until they are drained. The access gate in FIFO Access Gate sits on top of both fields: while it is closed, the FIFO neither captures nor reads out, whatever `FIFO_CTRL.ENABLE` holds.

#### FIFO Access Gate

Three FIFOs are readable through the register block, and each exposes entropy that must not leave the block in the field. The output FIFO holds the words that seed the DRBG. The two observe FIFOs hold entropy ahead of conditioning, which SP 800-90B treats as a validation-time interface that may be disabled in the field. `FIPS_LOCK` freezes their capture enables until reset, but only once it is set. A single access enable from the integration gates all three in hardware, whatever the register configuration holds.

Table 241 lists the three FIFOs and the condition under which each captures while access is enabled.

*Table 241. Readable FIFOs*

| FIFO | Read register | Captures while access is enabled |
|---|---|---|
| Conditioned output FIFO | `FIFO_RDATA` | `FIFO_CTRL.ENABLE` is set. The same words leave on `entropy_stream_data_o` whatever the FIFO does. |
| BIW observe FIFO | `BIW_OBS_RDATA` | `BIW_OBS_CTRL.RAW_ENABLE` is set and the boot health-test phase is done. |
| Raw-noise observe FIFO | `NOISE_OBS_RDATA` | `NOISE_OBS_CTRL.RAW_ENABLE` is set. The boot phase does not gate it. |

While access is enabled, a read of a data register pops one word and the capture conditions above apply unchanged. While access is disabled:

- A read of any of the three data registers returns zero and does not pop. The FIFO's level, pointers and error status are unaffected and stay readable.
- None of the three FIFOs captures, whatever its capture enable holds, so no entropy accumulates in a FIFO that cannot be drained and no overflow is reported. The output FIFO's write side is blocked; the seed stream towards the DRBG is not.
- The gate does not flush. Words already held are retained and become readable again once access is enabled.
- The capture-enable and `FIPS_LOCK` fields keep their normal write behaviour.

The gate is independent of `FIPS_LOCK`: the lock freezes the capture enables, the gate decides whether the FIFOs capture and read out at all, and both apply. The integration decides when the access enable is asserted; SEP derives it from the lifecycle controller, as the SEP TRNG chapter describes.

**NOTE** The access enable is specified ahead of the hardware. Until the RTL implements it, the three FIFOs are always accessible, subject only to `FIPS_LOCK`.

#### Reset

The entropy source `rst_ni` input resets its register block, health-test/FSM state, conditioning pipeline, counters, and FIFOs. Reserved `CTRL[0]` reads as zero and ignores writes.

At SEP integration level, `SEP_RESET_CTRL.SW_RESET_N.TRNG_SW_RST_N` requests a shared reset of the entropy source and DRBG. Hardware first drains and isolates the ESRC, CSRNG, and EDN AXI ports, then asserts the common reset. This is a full-IP reset: all entropy-source CSRs and `FIPS_LOCK` return to reset values. Post-mux EDN adapters and the fabric entropy

pool remain in the POR reset domain; they synchronously clear buffered entropy while the internal reset is active. After reset release, `MODULE_ENABLE` returning to one produces the normal enable pulse and restarts startup health testing before entropy is released.

### Debug Observation Pin

`DEBUG_CTRL.SELECT_SIGNAL` routes one of 256 internal signals to the `signal_monitor_o` output, and `DEBUG_CTRL.SELECT_FREQ_DIV` divides it down for external observation. Some selections expose entropy ahead of conditioning: the raw per-generator noise bits, the compressed 32-bit entropy stream, and the uncompressed per-generator entropy bytes.

Both fields are covered by `FIPS_LOCK`: once `FIPS_LOCK.LOCK` is set, software can no longer retarget the pin or change its divider, so a locked, certified part cannot route pre-conditioning entropy to the observation pin. The raw and BIW observe-FIFO capture enables (`NOISE_OBS_CTRL.RAW_ENABLE`, `BIW_OBS_CTRL.RAW_ENABLE`) are held under the lock the same way, and the FIFOs themselves sit behind the access gate in FIFO Access Gate. SP 800-90B treats this pre-conditioning observation as a validation-time interface that may be disabled in the field, which the lock and the gate enforce. As defense in depth, SEP also leaves `signal_monitor_o` unconnected and false-paths the port in synthesis; an integration that routes the pin off-chip still owns restricting access to the lifecycle or debug states in which observing pre-conditioning entropy is acceptable.

## Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

### Entropy Source Address Map

Entropy-source configuration, status, and data registers.

*Table 242. Entropy Source Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | COMPONENT_ID | R | Identifies the entropy-source register interface. |
| 0x4 | CTRL | RW | Controls entropy generation and conditioning. |
| 0xC | DEBUG_CTRL | RW | Controls the external debug observation pin. FIPS_LOCK.LOCK prevents software changes to the signal selection and divider. |
| 0x10 | INTR_STATUS | RW | Reports latched interrupt status; write one to clear a bit. |
| 0x14 | INTR_ENABLE | RW | Enables individual interrupt sources. |
| 0x18 | INTR_TEST | W | Triggers individual interrupt sources for testing. |
| 0x1C | SHA256_STATUS | R | Reports SHA-256 conditioner progress. |
| 0x20 | FIFO_CTRL | RW | Controls the 64-entry main entropy FIFO. |
| 0x24 | FIFO_STATUS | R | Reports main FIFO occupancy and pointers. |
| 0x28 | FIFO_RDATA | R | Provides the next main FIFO word. |
| 0x30 | HEALTH_TEST_CTRL | RW | Controls continuous entropy health tests. |
| 0x34 | HEALTH_TEST_WINDOW_SIZE | RW | Sets the APT and Markov window length. |
| 0x38 | MARKOV_TEST_PROB_THRESHOLDS | RW | Sets high and low Markov alternation-count thresholds. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x40 | HEALTH_TEST_STATUS | RW | Latches 32-bit stream health-test failures: repetition [0], APT high or low [3], Markov high [4], and Markov low [5]. Write one to clear. |
| 0x44 | REPETITION_TEST_COUNT | R | Reports the highest current repetition count. |
| 0x50 | APT_PATTERN_COUNT_1BIT | R | Reports the highest per-lane APT one-count in the current window. |
| 0x54 | APT_PATTERN_COUNT_2BIT | R | Reports the lowest per-lane APT one-count in the current window. |
| 0x60 | APT_PROPORTION_1BIT | RW | Sets the APT high one-count threshold. |
| 0x70 | APT_PROPORTION_LO | RW | Sets the APT low one-count threshold. |
| 0x80 | MARKOV_TEST_COUNTS_0 | R | Reports the highest and lowest per-lane alternation counts in the current Markov window. |
| 0x90 | RING_OSC_ENABLE | RW | Controls noise and sample-clock ring oscillators. |
| 0x94 | RING_OSC_TUNE | RW | Selects detuned ring-oscillator tap points. |
| 0x98 | RING_OSC_CTRL | RW | Selects each generator sample-clock source. |
| 0xA0 | DECORRELATOR_CTRL | RW | Controls generator decorrelation and byte cadence. |
| 0xA4 | DECORRELATOR_MASK | RW | Masks bits in every generator entropy byte. |
| 0xB4 | MAIN_SM_STATUS | RW | Reports the startup health-test gate state machine. |
| 0xC0 | GENERATOR_0_HEALTH_STATUS | RW | Latches generator 0 health-test failures; write one to clear. |
| 0xC4 | GENERATOR_1_HEALTH_STATUS | RW | Latches generator 1 health-test failures; write one to clear. |
| 0xC8 | GENERATOR_2_HEALTH_STATUS | RW | Latches generator 2 health-test failures; write one to clear. |
| 0xCC | GENERATOR_3_HEALTH_STATUS | RW | Latches generator 3 health-test failures; write one to clear. |
| 0xD0 | GENERATOR_4_HEALTH_STATUS | RW | Latches generator 4 health-test failures; write one to clear. |
| 0xD4 | GENERATOR_5_HEALTH_STATUS | RW | Latches generator 5 health-test failures; write one to clear. |
| 0xD8 | GENERATOR_6_HEALTH_STATUS | RW | Latches generator 6 health-test failures; write one to clear. |
| 0xDC | GENERATOR_7_HEALTH_STATUS | RW | Latches generator 7 health-test failures; write one to clear. |
| 0xE0 | GENERATOR_8_HEALTH_STATUS | RW | Latches generator 8 health-test failures; write one to clear. |
| 0xE4 | GENERATOR_9_HEALTH_STATUS | RW | Latches generator 9 health-test failures; write one to clear. |
| 0xE8 | GENERATOR_10_HEALTH_STATUS | RW | Latches generator 10 health-test failures; write one to clear. |
| 0xEC | GENERATOR_11_HEALTH_STATUS | RW | Latches generator 11 health-test failures; write one to clear. |
| 0x100 | GENERATOR_0_SAMPLE_CLK_CONFIG | RW | Controls generator 0 sample-clock division. |
| 0x104 | GENERATOR_1_SAMPLE_CLK_CONFIG | RW | Controls generator 1 sample-clock division. |
| 0x108 | GENERATOR_2_SAMPLE_CLK_CONFIG | RW | Controls generator 2 sample-clock division. |
| 0x10C | GENERATOR_3_SAMPLE_CLK_CONFIG | RW | Controls generator 3 sample-clock division. |
| 0x110 | GENERATOR_4_SAMPLE_CLK_CONFIG | RW | Controls generator 4 sample-clock division. |
| 0x114 | GENERATOR_5_SAMPLE_CLK_CONFIG | RW | Controls generator 5 sample-clock division. |
| 0x118 | GENERATOR_6_SAMPLE_CLK_CONFIG | RW | Controls generator 6 sample-clock division. |
| 0x11C | GENERATOR_7_SAMPLE_CLK_CONFIG | RW | Controls generator 7 sample-clock division. |
| 0x120 | GENERATOR_8_SAMPLE_CLK_CONFIG | RW | Controls generator 8 sample-clock division. |
| 0x124 | GENERATOR_9_SAMPLE_CLK_CONFIG | RW | Controls generator 9 sample-clock division. |
| 0x128 | GENERATOR_10_SAMPLE_CLK_CONFIG | RW | Controls generator 10 sample-clock division. |
| 0x12C | GENERATOR_11_SAMPLE_CLK_CONFIG | RW | Controls generator 11 sample-clock division. |
| 0x130 | HT_WATERMARK_NUM | RW | Selects the counter and direction for the shared health-test watermark. Changing the selection preserves the current watermark. |
| 0x134 | HT_WATERMARK | R | Reports one selector-dependent watermark. A CTRL.MODULE_ENABLE rising edge arms high modes to zero and low modes to 0xFFFF. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x138 | REPCNT_TOTAL_FAILS | R | Counts repetition-test failures since the last CTRL.MODULE_ENABLE rising edge. |
| 0x13C | APT_HI_TOTAL_FAILS | R | Counts APT high-threshold failures since the last CTRL.MODULE_ENABLE rising edge. |
| 0x140 | APT_LO_TOTAL_FAILS | R | Counts APT low-threshold failures since the last CTRL.MODULE_ENABLE rising edge. |
| 0x144 | MARKOV_HI_TOTAL_FAILS | R | Counts Markov high-threshold failures since the last CTRL.MODULE_ENABLE rising edge. |
| 0x148 | MARKOV_LO_TOTAL_FAILS | R | Counts Markov low-threshold failures since the last CTRL.MODULE_ENABLE rising edge. |
| 0x14C | ALERT_SUMMARY_FAIL_COUNTS | R | Counts consecutive health-test windows containing any failure and clears after an accepted passing window. |
| 0x150 | ALERT_FAIL_COUNTS | R | Reports saturating per-test failure counts for the current alert sequence. |
| 0x154 | FIPS_LOCK | RW | Locks entropy generation, conditioning, and health-test configuration against software writes until hardware reset. |
| 0x158 | ALERT_THRESHOLD | RW | Sets the failing-window count that enters the persistent-failure alert path. |
| 0x15C | MIN_ENTROPY_H | RW | Stores assessed per-sample min-entropy in unsigned Q4.4 format for advisory threshold calculation. |
| 0x160 | RECOMMENDED_THRESHOLDS | R | Reports advisory repetition and APT thresholds derived from MIN_ENTROPY_H. |
| 0x164 | BIW_OBS_CTRL | RW | Controls copy-only capture of the post-BIW, pre-SHA stream. |
| 0x168 | BIW_OBS_STATUS | R | Reports BIW observe FIFO occupancy and pointers. |
| 0x16C | BIW_OBS_RDATA | R | Provides the next post-BIW, pre-SHA observe word. |
| 0x170 | NOISE_OBS_CTRL | RW | Controls copy-only capture of one pre-decorrelator generator lane. |
| 0x174 | NOISE_OBS_STATUS | R | Reports raw-noise observe FIFO occupancy and pointers. |
| 0x178 | NOISE_OBS_RDATA | R | Provides the next packed raw-noise observe word. |

### COMPONENT_ID

*Table 243. entropy_source.COMPONENT_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `MAJOR_VERSION` | R | 0x0 | Major interface version. |
| 27:24 | `MINOR_VERSION` | R | 0x1 | Minor interface version. |
| 23:16 | `Reserved` | - | - | Reserved |
| 15:0 | `NAME` | R | 0x1 | Entropy-source component identifier. |

### CTRL

*Table 244. entropy_source.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `SHA256_WHITENING_ENABLE` | RW | 0x1 | Enables SHA-256 conditioning of each 512-bit BIW input block into a 256-bit digest. When clear, BIW words pass directly to the FIFO. FIPS_LOCK.LOCK prevents software changes. |
| 27:26 | `Reserved` | - | - | Reserved |
| 25:16 | `DOWNSAMPLE_RATE` | RW | 0x0 | Selects every (N+1)th decorrelator output while BIW extraction is bypassed. FIPS_LOCK.LOCK prevents software changes. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `BYPASS_ENTROPY_COMPRESSOR` | RW | 0x0 | Bypasses BIW extraction and writes the twelve generator bytes to the FIFO as three 32-bit words, word 0 first, with generator 4n+k in bits [8k+7:8k] of word n. FIPS_LOCK.LOCK prevents software changes. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `AUTOTUNE_ENABLE` | RW | 0x0 | Enables per-generator automatic detuning after a lane health-test failure. FIPS_LOCK.LOCK prevents software changes. |
| 3:2 | `Reserved` | - | - | Reserved |
| 1 | `MODULE_ENABLE` | RW | 0x1 | Enables startup health testing and entropy output. Clearing this field holds the main state machine idle. FIPS_LOCK.LOCK prevents software changes. |
| 0 | `RSVD0` | R | 0x0 | Reserved; reads zero and ignores writes. |

### DEBUG_CTRL

*Table 245. entropy_source.DEBUG_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10:8 | `SELECT_FREQ_DIV` | RW | 0x0 | Selects observation division by 1, 2, 4, 8, 16, 32, 64, or 128. FIPS_LOCK.LOCK prevents software changes. |
| 7:0 | `SELECT_SIGNAL` | RW | 0x0 | Selects the observed signal: generator noise bits [11:0], sample clocks [27:16], BIW data [63:32], FIFO input [95:64], or generator bytes [223:128]. Other selections drive the divider input low. FIPS_LOCK.LOCK prevents software changes. |

### INTR_STATUS

*Table 246. entropy_source.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `NOISE_OBS_OVERFLOW` | RW1C | 0x0 | Indicates that a raw-noise observe FIFO word was discarded while full. Write one to clear. |
| 27:25 | `Reserved` | - | - | Reserved |
| 24 | `BIW_OBS_OVERFLOW` | RW1C | 0x0 | Indicates that a BIW observe FIFO word was discarded while full. Write one to clear. |
| 23:21 | `Reserved` | - | - | Reserved |
| 20 | `AUTOTUNE_FAIL` | RW1C | 0x0 | Indicates a per-generator health-test failure while automatic detuning is enabled. Write one to clear. |
| 19:17 | `Reserved` | - | - | Reserved |
| 16 | `PERSISTENT_FAILURE` | RW1C | 0x0 | Indicates a persistent health-test failure or health-test counter error. Write one to clear. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12 | `FIFO_UNDERFLOW` | RW1C | 0x0 | Indicates that software read the empty main FIFO. Write one to clear. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `FIFO_OVERFLOW` | RW1C | 0x0 | Indicates that a main FIFO write was discarded while full. Write one to clear. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `FIFO_ERROR` | RW1C | 0x0 | Indicates a FIFO integrity error. Write one to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `HEALTH_TEST_FAILED` | RW1C | 0x0 | Indicates a failure in the 32-bit entropy-stream health tests. Write one to clear. |

### INTR_ENABLE

*Table 247. entropy_source.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `NOISE_OBS_OVERFLOW` | RW | 0x0 | Enables the raw-noise observe FIFO overflow interrupt. |
| 27:25 | `Reserved` | - | - | Reserved |
| 24 | `BIW_OBS_OVERFLOW` | RW | 0x0 | Enables the BIW observe FIFO overflow interrupt. |
| 23:21 | `Reserved` | - | - | Reserved |
| 20 | `AUTOTUNE_FAIL` | RW | 0x0 | Enables the automatic-detuning failure interrupt. |
| 19:17 | `Reserved` | - | - | Reserved |
| 16 | `PERSISTENT_FAILURE` | RW | 0x0 | Enables the persistent-failure interrupt. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12 | `FIFO_UNDERFLOW` | RW | 0x0 | Enables the main FIFO underflow interrupt. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `FIFO_OVERFLOW` | RW | 0x0 | Enables the main FIFO overflow interrupt. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `FIFO_ERROR` | RW | 0x0 | Enables the FIFO integrity-error interrupt. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `HEALTH_TEST_FAILED` | RW | 0x0 | Enables the health-test failure interrupt. |

### INTR_TEST

*Table 248. entropy_source.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `NOISE_OBS_OVERFLOW` | W 1P | 0x0 | Write one to pulse the raw-noise observe FIFO overflow interrupt source. |
| 27:25 | `Reserved` | - | - | Reserved |
| 24 | `BIW_OBS_OVERFLOW` | W 1P | 0x0 | Write one to pulse the BIW observe FIFO overflow interrupt source. |
| 23:21 | `Reserved` | - | - | Reserved |
| 20 | `AUTOTUNE_FAIL` | W 1P | 0x0 | Write one to pulse the automatic-detuning failure interrupt source. |
| 19:17 | `Reserved` | - | - | Reserved |
| 16 | `PERSISTENT_FAILURE` | W 1P | 0x0 | Write one to pulse the persistent-failure interrupt source. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12 | `FIFO_UNDERFLOW` | W 1P | 0x0 | Write one to pulse the main FIFO underflow interrupt source. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `FIFO_OVERFLOW` | W 1P | 0x0 | Write one to pulse the main FIFO overflow interrupt source. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `FIFO_ERROR` | W 1P | 0x0 | Write one to pulse the FIFO integrity-error interrupt source. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `HEALTH_TEST_FAILED` | W 1P | 0x0 | Write one to pulse the health-test failure interrupt source. |

### SHA256_STATUS

*Table 249. entropy_source.SHA256_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `Reserved` | - | - | Reserved |
| 11:8 | `OUTPUT_COUNT` | R | 0x0 | Number of 32-bit digest words remaining for output. Reports 8 when a digest becomes available, decrements through 0 as words are accepted, and remains 0 while idle or bypassed. |
| 7:4 | `INPUT_COUNT` | R | 0x0 | Number of 32-bit words currently accumulated in the 512-bit input block. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `BUSY` | R | 0x0 | Indicates that SHA-256 processing or digest output is active. |

### FIFO_CTRL

*Table 250. entropy_source.FIFO_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `ENTROPY_CHURN_ENABLE` | RW | 0x0 | XORs each incoming word with the FIFO entry 32 positions ahead of the write pointer. FIPS_LOCK.LOCK prevents software changes. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `ENABLE` | RW | 0x1 | Enables FIFO writes. Reads remain available while writes are disabled. Words offered to the FIFO are also the DRBG seed stream, which this field does not gate. FIPS_LOCK.LOCK prevents software changes. |

### FIFO_STATUS

*Table 251. entropy_source.FIFO_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:22 | `Reserved` | - | - | Reserved |
| 21:16 | `RPTR` | R | 0x0 | Current FIFO read pointer. |
| 15:14 | `Reserved` | - | - | Reserved |
| 13:8 | `WPTR` | R | 0x0 | Current FIFO write pointer. |
| 7 | `Reserved` | - | - | Reserved |
| 6:0 | `LEVEL` | R | 0x0 | Number of valid 32-bit words in the FIFO. |

### FIFO_RDATA

*Table 252. entropy_source.FIFO_RDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `RDATA` | R RMOD | 0x0 | Returns and removes the next FIFO word. Reading an empty FIFO sets INTR_STATUS.FIFO_UNDERFLOW and returns undefined data. |

### HEALTH_TEST_CTRL

*Table 253. entropy_source.HEALTH_TEST_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:8 | `REPETITION_LIMIT` | RW | 0x19 | Maximum allowed per-lane repetition count before the test fails. FIPS_LOCK.LOCK prevents software changes. |
| 7:3 | `Reserved` | - | - | Reserved |
| 2:0 | `ENABLE` | RW | 0x7 | Enables the repetition-count, adaptive-proportion, and Markov tests in bits 0, 1, and 2. Disabling a test clears its state. |

### HEALTH_TEST_WINDOW_SIZE

*Table 254. entropy_source.HEALTH_TEST_WINDOW_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `SIZE` | RW | 0x800 | Number of valid entropy words per APT and Markov window. FIPS_LOCK.LOCK requires a value of at least 1024 and prevents software changes. |

### MARKOV_TEST_PROB_THRESHOLDS

*Table 255. entropy_source.MARKOV_TEST_PROB_THRESHOLDS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `PROB_10_THRESHOLD` | RW | 0x64 | Minimum allowed per-lane alternation count at the window boundary. FIPS_LOCK.LOCK prevents software changes. |
| 15:0 | `PROB_01_THRESHOLD` | RW | 0x4B0 | Maximum allowed per-lane alternation count at the window boundary. FIPS_LOCK.LOCK prevents software changes. |

### HEALTH_TEST_STATUS

*Table 256. entropy_source.HEALTH_TEST_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `HEALTH_STATUS` | RW1C | 0x0 | Latched health-test failure bits; write one to clear. |

### REPETITION_TEST_COUNT

*Table 257. entropy_source.REPETITION_TEST_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `REPETITION_COUNT` | R | 0x0 | Highest current repetition count among the 32 entropy lanes. |

### APT_PATTERN_COUNT_1BIT

*Table 258. entropy_source.APT_PATTERN_COUNT_1BIT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `PATTERN_COUNT` | R | 0x0 | Highest current one-count among the 32 entropy lanes. |

### APT_PATTERN_COUNT_2BIT

*Table 259. entropy_source.APT_PATTERN_COUNT_2BIT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `PATTERN_COUNT` | R | 0x0 | Lowest current one-count among the 32 entropy lanes. |

### APT_PROPORTION_1BIT

*Table 260. entropy_source.APT_PROPORTION_1BIT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `LIMIT` | RW | 0x4B0 | Maximum allowed per-lane one-count at the window boundary. FIPS_LOCK.LOCK prevents software changes. |

### APT_PROPORTION_LO

*Table 261. entropy_source.APT_PROPORTION_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `LIMIT` | RW | 0x350 | Minimum allowed per-lane one-count at the window boundary. Zero disables the low-side test. FIPS_LOCK.LOCK prevents software changes. |

### MARKOV_TEST_COUNTS_0

*Table 262. entropy_source.MARKOV_TEST_COUNTS_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `COUNT_10` | R | 0x0 | Lowest current per-lane alternation count. |
| 15:0 | `COUNT_01` | R | 0x0 | Highest current per-lane alternation count. |

### RING_OSC_ENABLE

*Table 263. entropy_source.RING_OSC_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:12 | `SAMPLE_CLK_ENABLE` | RW | 0xFFF | Enables the shared sample-clock ring oscillator when any bit is set. FIPS_LOCK.LOCK prevents software changes. |
| 11:0 | `ENABLE` | RW | 0xFFF | Enables each generator noise ring oscillator. FIPS_LOCK.LOCK prevents software changes. |

### RING_OSC_TUNE

*Table 264. entropy_source.RING_OSC_TUNE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:12 | `SAMPLE_CLK_DETUNE` | RW | 0x0 | Selects the full-length (longer) shared sample-clock oscillator tap, lowering its frequency, when any bit is set. FIPS_LOCK.LOCK prevents software changes. |
| 11:0 | `DETUNE` | RW | 0x0 | Selects the full-length (longer) tap for each noise ring oscillator, lowering its frequency, while automatic tuning is disabled; the reset value (clear) selects the shorter tap. FIPS_LOCK.LOCK prevents software changes. |

### RING_OSC_CTRL

*Table 265. entropy_source.RING_OSC_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `Reserved` | - | - | Reserved |
| 11:0 | `SAMPLE_CLK_SELECT` | RW | 0xFFF | Selects the shared internal sample clock when set and the external sample clock when clear. FIPS_LOCK.LOCK prevents software changes. |

### DECORRELATOR_CTRL

*Table 266. entropy_source.DECORRELATOR_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `SAMPLE_CLK_DIV` | RW | 0x3F | Sets the generator byte-output period to this value plus one sample-clock cycles. FIPS_LOCK.LOCK prevents software changes. |
| 11:0 | `BYPASS` | RW | 0x0 | Bypasses the LFSR feedback path independently for each generator. FIPS_LOCK.LOCK prevents software changes. |

### DECORRELATOR_MASK

*Table 267. entropy_source.DECORRELATOR_MASK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `ENTROPY_BYTE_MASK` | RW | 0xFF | Enables corresponding entropy-byte bits when set. FIPS_LOCK.LOCK prevents software changes. |

### MAIN_SM_STATUS

*Table 268. entropy_source.MAIN_SM_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:14 | `Reserved` | - | - | Reserved |
| 13 | `ALERT_CNTR_CLR_OK` | R | 0x0 | Indicates that a passing window may clear alert failure counters. |
| 12 | `BOOT_PHASE_DONE` | R | 0x0 | Indicates successful completion of startup health testing and permits entropy output. |
| 11 | `ERR` | RW1C | 0x0 | Latches an illegal state or health-test counter integrity error. Write one or reset the block to clear. |
| 10 | `ALERT` | RW1C | 0x0 | Latches entry into the persistent-failure alert path, which halts entropy output. Write one or reset the block to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 9 | IDLE | R | 0x0 | Indicates that the state machine is idle. |
| 8:0 | STATE | R | 0x0 | Current sparse-encoded state-machine state. |

### GENERATOR_0_HEALTH_STATUS

*Table 269. entropy_source.GENERATOR_0_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 0 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_1_HEALTH_STATUS

*Table 270. entropy_source.GENERATOR_1_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 1 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_2_HEALTH_STATUS

*Table 271. entropy_source.GENERATOR_2_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 2 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_3_HEALTH_STATUS

*Table 272. entropy_source.GENERATOR_3_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 3 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_4_HEALTH_STATUS

*Table 273. entropy_source.GENERATOR_4_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 4 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_5_HEALTH_STATUS

*Table 274. entropy_source.GENERATOR_5_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | STATUS | RW1C | 0x0 | Generator 5 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_6_HEALTH_STATUS

*Table 275. entropy_source.GENERATOR_6_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 6 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_7_HEALTH_STATUS

*Table 276. entropy_source.GENERATOR_7_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 7 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_8_HEALTH_STATUS

*Table 277. entropy_source.GENERATOR_8_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 8 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_9_HEALTH_STATUS

*Table 278. entropy_source.GENERATOR_9_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 9 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_10_HEALTH_STATUS

*Table 279. entropy_source.GENERATOR_10_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 10 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_11_HEALTH_STATUS

*Table 280. entropy_source.GENERATOR_11_HEALTH_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `STATUS` | RW1C | 0x0 | Generator 11 failure bits use the HEALTH_TEST_STATUS allocation. |

### GENERATOR_0_SAMPLE_CLK_CONFIG

*Table 281. entropy_source.GENERATOR_0_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x0 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_1_SAMPLE_CLK_CONFIG

*Table 282. entropy_source.GENERATOR_1_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x0 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_2_SAMPLE_CLK_CONFIG

*Table 283. entropy_source.GENERATOR_2_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x1 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_3_SAMPLE_CLK_CONFIG

*Table 284. entropy_source.GENERATOR_3_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x1 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_4_SAMPLE_CLK_CONFIG

*Table 285. entropy_source.GENERATOR_4_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x2 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_5_SAMPLE_CLK_CONFIG

*Table 286. entropy_source.GENERATOR_5_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x2 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_6_SAMPLE_CLK_CONFIG

*Table 287. entropy_source.GENERATOR_6_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x0 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_7_SAMPLE_CLK_CONFIG

*Table 288. entropy_source.GENERATOR_7_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x0 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_8_SAMPLE_CLK_CONFIG

*Table 289. entropy_source.GENERATOR_8_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x1 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_9_SAMPLE_CLK_CONFIG

*Table 290. entropy_source.GENERATOR_9_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x1 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_10_SAMPLE_CLK_CONFIG

*Table 291. entropy_source.GENERATOR_10_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x2 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### GENERATOR_11_SAMPLE_CLK_CONFIG

*Table 292. entropy_source.GENERATOR_11_SAMPLE_CLK_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `SAMPLE_CLK_DIVIDE` | RW | 0x2 | Selects division by 1, 2, 4, 8, 16, or 32. FIPS_LOCK.LOCK prevents software changes. |

### HT_WATERMARK_NUM

*Table 293. entropy_source.HT_WATERMARK_NUM fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3:0 | `WATERMARK_NUM` | RW | 0x0 | Selects the watermark source. Unsupported values select REPCNT_HI. |

### HT_WATERMARK

*Table 294. entropy_source.HT_WATERMARK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `WATERMARK_VALUE` | R | 0x0 | Current shared watermark value. |

### REPCNT_TOTAL_FAILS

*Table 295. entropy_source.REPCNT_TOTAL_FAILS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FAIL_COUNT` | R | 0x0 | Total repetition-test failure pulses. |

### APT_HI_TOTAL_FAILS

*Table 296. entropy_source.APT_HI_TOTAL_FAILS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FAIL_COUNT` | R | 0x0 | Total APT high-threshold failure pulses. |

### APT_LO_TOTAL_FAILS

*Table 297. entropy_source.APT_LO_TOTAL_FAILS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FAIL_COUNT` | R | 0x0 | Total APT low-threshold failure pulses. |

### MARKOV_HI_TOTAL_FAILS

*Table 298. entropy_source.MARKOV_HI_TOTAL_FAILS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FAIL_COUNT` | R | 0x0 | Total Markov high-threshold failure pulses. |

### MARKOV_LO_TOTAL_FAILS

*Table 299. entropy_source.MARKOV_LO_TOTAL_FAILS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FAIL_COUNT` | R | 0x0 | Total Markov low-threshold failure pulses. |

### ALERT_SUMMARY_FAIL_COUNTS

*Table 300. entropy_source.ALERT_SUMMARY_FAIL_COUNTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `ANY_FAIL_COUNT` | R | 0x0 | Number of consecutive failing health-test windows. |

### ALERT_FAIL_COUNTS

*Table 301. entropy_source.ALERT_FAIL_COUNTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:16 | `REPCNT_FAIL_COUNT` | R | 0x0 | Saturating repetition-test failure count. |
| 15:12 | `MARKOV_HI_FAIL_COUNT` | R | 0x0 | Saturating Markov high-threshold failure count. |
| 11:8 | `MARKOV_LO_FAIL_COUNT` | R | 0x0 | Saturating Markov low-threshold failure count. |
| 7:4 | `APT_HI_FAIL_COUNT` | R | 0x0 | Saturating APT high-threshold failure count. |
| 3:0 | `APT_LO_FAIL_COUNT` | R | 0x0 | Saturating APT low-threshold failure count. |

### FIPS_LOCK

*Table 302. entropy_source.FIPS_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `LOCK` | RW1S | 0x0 | Write one to set the configuration lock. Only hardware reset clears it. |

### ALERT_THRESHOLD

*Table 303. entropy_source.ALERT_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `THRESHOLD` | RW | 0x4 | Number of consecutive failing windows that triggers the alert; zero disables this trigger. FIPS_LOCK.LOCK prevents software changes. |

### MIN_ENTROPY_H

*Table 304. entropy_source.MIN_ENTROPY_H fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `H` | RW | 0xC | Assessed min-entropy in bits per sample. FIPS_LOCK.LOCK prevents software changes. |

### RECOMMENDED_THRESHOLDS

*Table 305. entropy_source.RECOMMENDED_THRESHOLDS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `APT_LIMIT` | R | 0x0 | Advisory APT high cutoff derived for a 1024-sample window. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:0 | RCT_LIMIT | R | 0x0 | Advisory repetition-count cutoff C = 1 + ceil(20 / H). |

### BIW_OBS_CTRL

*Table 306. entropy_source.BIW_OBS_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | - | - | Reserved |
| 0 | RAW_ENABLE | RW | 0x0 | Enables capture after startup health testing completes. FIPS_LOCK.LOCK prevents software changes. |

### BIW_OBS_STATUS

*Table 307. entropy_source.BIW_OBS_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:22 | Reserved | - | - | Reserved |
| 21:16 | RPTR | R | 0x0 | Current BIW observe FIFO read pointer. |
| 15:14 | Reserved | - | - | Reserved |
| 13:8 | WPTR | R | 0x0 | Current BIW observe FIFO write pointer. |
| 7 | Reserved | - | - | Reserved |
| 6:0 | LEVEL | R | 0x0 | Number of valid 32-bit words in the BIW observe FIFO. |

### BIW_OBS_RDATA

*Table 308. entropy_source.BIW_OBS_RDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RDATA | R RMOD | 0x0 | Returns and removes the next BIW observe FIFO word. |

### NOISE_OBS_CTRL

*Table 309. entropy_source.NOISE_OBS_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:4 | LANE_SEL | RW | 0x0 | Selects generator lane 0 through 11; other values select lane 0. Changing lanes restarts packing and flushes buffered words. |
| 3:2 | Reserved | - | - | Reserved |
| 1 | FLUSH | W 1P | 0x0 | Write one to discard buffered raw-noise words. Changing LANE_SEL also flushes the FIFO. |
| 0 | RAW_ENABLE | RW | 0x0 | Enables LSB-first packing of selected-lane samples into 32-bit observe words. FIPS_LOCK.LOCK prevents software changes. |

### NOISE_OBS_STATUS

*Table 310. entropy_source.NOISE_OBS_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:22 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 21:16 | RPTR | R | 0x0 | Current raw-noise observe FIFO read pointer. |
| 15:14 | Reserved | - | - | Reserved |
| 13:8 | WPTR | R | 0x0 | Current raw-noise observe FIFO write pointer. |
| 7 | Reserved | - | - | Reserved |
| 6:0 | LEVEL | R | 0x0 | Number of valid 32-bit words in the raw-noise observe FIFO. |

#### NOISE_OBS_RDATA

*Table 311. entropy_source.NOISE_OBS_RDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RDATA | R RMOD | 0x0 | Returns and removes 32 LSB-first pre-decorrelator samples from the selected lane. |

### Entropy Pool

The entropy pool (`sep_entropy_fifo`) is a read-only 64-bit AXI aperture through which software drains conditioned entropy. It is a top-level block in `sep`, reached as a target on the SEP local crossbar; see the Memory Map for its address window. The pool is filled autonomously by a native EDN endpoint on a separate hardware path, not through the crossbar; see the TRNG and DRBG chapter for the fill path and the entropy it carries.

#### Access Policy

- A read of `DATA` pops one 64-bit conditioned word. Reading an empty pool returns `SLVERR` rather than `OKAY`, so an empty pool is distinguishable from a genuine zero word.
- `STATUS` and `IRQ_CAUSE` are non-destructive reads. `STATUS` reports pool occupancy (`fifo_level`), the low-watermark condition (`pool_low`), an EDN fill stall (`fill_stall`) and a FIFO integrity fault (`pool_error`); the latter three are also raised to the PIC (see the Interrupts chapter).
- The pool is read-only. A write to any offset terminates with `BRESP=SLVERR` and changes no state.
- All three registers are 64-bit access-width (`STATUS`, `IRQ_CAUSE` and `DATA` each occupy one 8-byte slot) and are read with a single 64-bit access. A 32-bit read of a register's upper word (offset +4) is not decoded: it returns `SLVERR` with zero data, and for `DATA` it does not pop the pool. Read the full 64-bit word to obtain a register's value. The exact decode response codes are tabulated with the aperture in the Memory Map.
- The aperture accepts single-beat accesses only (Single-Beat Register Access). An access to an unmapped offset within the aperture likewise returns `SLVERR` with zero data.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### SEP Entropy Pool

Read-only entropy pool FIFO; DATA reads drain conditioned entropy supplied by the native EDN, and writes return

SLVERR.

*Table 312. SEP Entropy Pool register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | STATUS | R | - |
| 0x8 | IRQ_CAUSE | R | - |
| 0x10 | DATA | R | - |

### STATUS

*Table 313. sep_entropy_pool.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:9 | `Reserved` | - | - | Reserved |
| 8 | `pool_error` | R | - | The pool detected a FIFO integrity error |
| 7 | `fill_stall` | R | - | The EDN fill request exceeded the configured stall threshold |
| 6 | `pool_low` | R | - | The pool occupancy is below the configured low watermark |
| 5:0 | `fifo_level` | R | - | Current number of available 64-bit entropy words |

### IRQ_CAUSE

*Table 314. sep_entropy_pool.IRQ_CAUSE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `pool_error` | R | - | - |
| 1 | `fill_stall` | R | - | - |
| 0 | `pool_low` | R | - | - |

### DATA

*Table 315. sep_entropy_pool.DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `entropy` | R | - | Pops one conditioned entropy word; an empty pool returns SLVERR |

### AES Cipher

The module is based on the OpenTitan AES design.

Key sizes: 128, 192 and 256 Modes: ECB, CBC, CFB, OFB, CTR, CTR-GCM

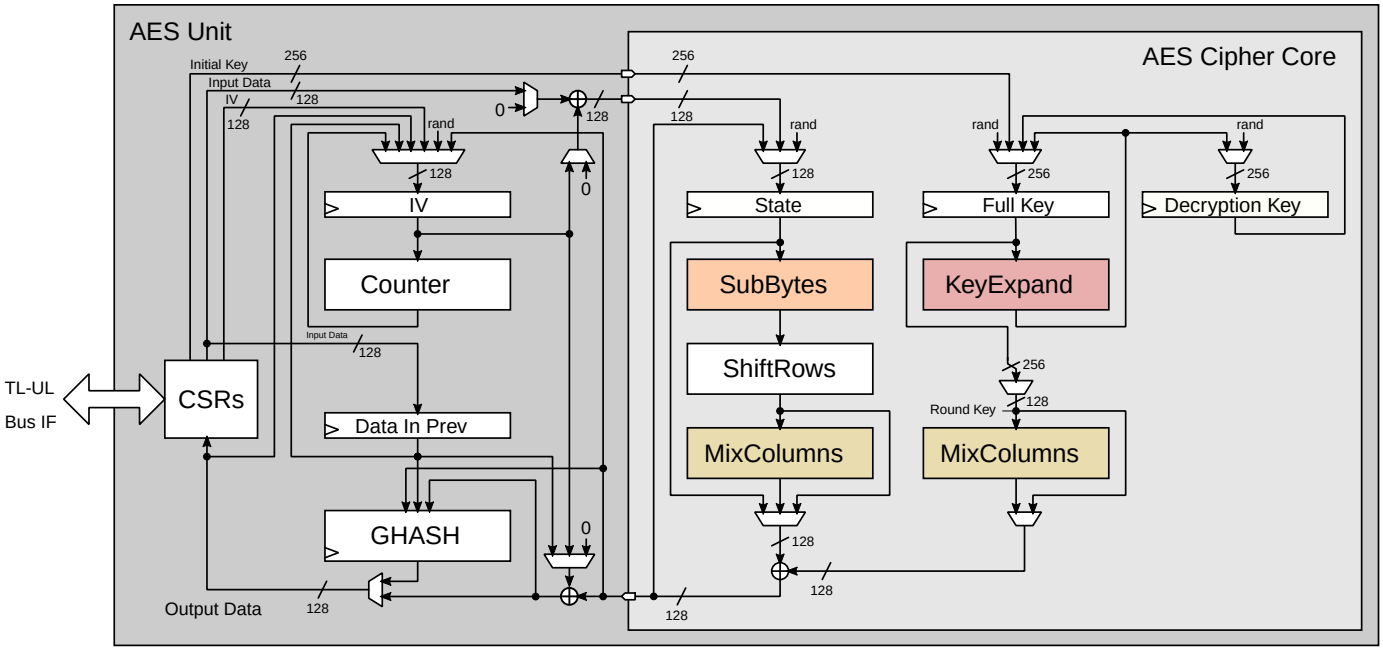


*Figure 37. OpenTitan AES core reference architecture*

Figure 37 shows the upstream core. The OCAH bus, key-delivery, and entropy connections are described in the integration details below.

### OCAH Modifications

- AXI4-lite key bus slave interface replaces side loading interface
- AXI4-lite control/status bus slave interface

The AES host path follows the standard conversion and isolation sequence in the crypto interconnect (64-bit AXI → 32-bit AXI → AXI-Lite → isolate), without address adjustment or cache-attribute patching. `aes_wrapper` bridges the isolated path to TL-UL. EDN requests are served by the shared `drbg_axis_edn_adapter` instance `u_axis_edn_crypto_s3c_scan`, which converts the post-CSRNG DRBG AXI-Stream feed for the crypto sink domain into AES's native-protocol EDN response. The adapter forwards the per-beat FIPS bit from the stream's `tuser` sideband into `edn_fips`; AES itself does not consume the flag. The AES core uses `rst_ni` for both the main and shadow reset domains. Life-cycle escalation is tied to its inactive state. AES occupies alert indices `[4:3]` in OpenTitan Alert Termination and exports an `idle_o` status output but no interrupt.

The Key Manager delivers a 256-bit AES key as eight words per share, in the format given in Key Manager Key Delivery. In AES, `CTRL_SHADOWED.sideload` selects the delivered key. While that bit is 0 the engine uses its software key registers and the delivered key has no effect, and no error is reported. `CTRL_SHADOWED.key_len` is independent of the delivery path: only its 256-bit encoding consumes the whole delivered key, and a shorter setting consumes only part of it. AES carries the two shares into the cipher datapath as a masked pair and does not combine them on the delivery path.

Consult the OpenTitan AES documentation for the authoritative core functionality of the IP. The Programmer's Guide gives the programming guidance for this block. The register map follows below.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

#### AES Accelerator

AES encryption and decryption engine with SCA and FI countermeasures

| | |
|---|---|
| **NOTE** | **Register Arrays:**<br>• **KEY_SHARE0[8]**: 8 registers, base 0x4, stride 0x4<br>• **KEY_SHARE1[8]**: 8 registers, base 0x24, stride 0x4<br>• **IV[4]**: 4 registers, base 0x44, stride 0x4<br>• **DATA_IN[4]**: 4 registers, base 0x54, stride 0x4<br>• **DATA_OUT[4]**: 4 registers, base 0x64, stride 0x4 |

*Table 316. AES Accelerator register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | ALERT_TEST | W | - |
| 0x4 - 0x20 | KEY_SHARE0[8] | W | - |
| 0x24 - 0x40 | KEY_SHARE1[8] | W | - |
| 0x44 - 0x50 | IV[4] | RW | - |
| 0x54 - 0x60 | DATA_IN[4] | W | - |
| 0x64 - 0x70 | DATA_OUT[4] | R | - |
| 0x74 | CTRL_SHADOWED | RW | - |
| 0x78 | CTRL_AUX_SHADOWED | RW | - |
| 0x7C | CTRL_AUX_REGWEN | RW | - |
| 0x80 | TRIGGER | W | - |
| 0x84 | STATUS | R | - |
| 0x88 | CTRL_GCM_SHADOWED | RW | - |

#### ALERT_TEST

*Table 317. aes.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `FATAL_FAULT` | W | 0x0 | Write 1 to trigger one alert event of this kind. |
| 0 | `RECOV_CTRL_UPDATE_ERR` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

#### KEY_SHARE0[8]

*Table 318. aes.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `KEY_SHARE0` | W | 0x0 | Initial Key Share 0 |

### KEY_SHARE1[8]

*Table 319. aes.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | KEY_SHARE1 | W | 0x0 | Initial Key Share 1 |

### IV[4]

*Table 320. aes.IV[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | IV | RW | 0x0 | Initialization Vector |

### DATA_IN[4]

*Table 321. aes.DATA_IN[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA_IN | W | 0x0 | Input Data |

### DATA_OUT[4]

*Table 322. aes.DATA_OUT[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA_OUT | R | 0x0 | Output Data |

### CTRL_SHADOWED

*Table 323. aes.CTRL_SHADOWED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | Reserved | - | - | Reserved |
| 15 | MANUAL_OPERATION | RW | 0x0 | Controls whether the AES unit is operated in normal/automatic mode (0) or fully manual mode (1). In automatic mode (0), the AES unit automatically i) starts to encrypt/decrypt when it receives new input data, and ii) stalls during the last encryption/decryption cycle if the previous output data has not yet been read. This is the most efficient mode to operate in. Note that the corresponding status tracking is automatically cleared upon a write to the Control Register. In manual mode (1), the AES unit i) only starts to encrypt/decrypt after receiving a start trigger (see Trigger Register), and ii) overwrites previous output data irrespective of whether it has been read out or not. This mode is useful if software needs full control over the AES unit. |
| 14:12 | PRNG_RESEED_RATE | RW | 0x1 | 3-bit one-hot field to control the reseeding rate of the internal pseudo-random number generator (PRNG) used for masking. Invalid input values, i.e., values with multiple bits set and value 3'b000 are mapped to the highest reseeding rate PER_1 (3'b001). |
| 11 | SIDELOAD | RW | 0x0 | Controls whether the AES unit uses the key provided by the key manager via key sideload interface (1) or the key provided by software via Initial Key Registers KEY_SHARE1_0 - KEY_SHARE1_7 (0). |
| 10:8 | KEY_LEN | RW | 0x1 | 3-bit one-hot field to select AES key length. Invalid input values, i.e., values with multiple bits set, value 3'b000, and value 3'b010 in case 192-bit keys are not supported (because disabled at compile time) are mapped to AES_256 (3'b100). |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:2 | `MODE` | RW | 0x3F | 6-bit one-hot field to select AES block cipher mode. Invalid input values, i.e., values with multiple bits set and value 6'b00_0000, are mapped to AES_NONE (6'b11_1111). |
| 1:0 | `OPERATION` | RW | 0x1 | 2-bit one-hot field to select the operation of AES unit. Invalid input values, i.e., values with multiple bits set and value 2'b00, are mapped to AES_ENC (2'b01). |

### CTRL_AUX_SHADOWED

*Table 324. aes.CTRL_AUX_SHADOWED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `FORCE_MASKS` | RW | 0x0 | Allow the internal masking PRNG to advance (0) or force its internal state (1) leading to constant masks. Setting all masks to constant value can be useful when performing SCA. To completely disable the masking, the second key share (KEY_SHARE1_0 - KEY_SHARE1_7) must be zero as well. In addition, a special seed needs to be loaded into the masking PRNG using the EDN interface. Only applicable if both the Masking parameter and the SecAllowForcingMasks parameter are set to one. |
| 0 | `KEY_TOUCH_FORCES_RESEED` | RW | 0x1 | Controls whether providing a new key triggers the reseeding of internal pseudo-random number generators used for clearing and masking (1) or not (0). |

### CTRL_AUX_REGWEN

*Table 325. aes.CTRL_AUX_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `CTRL_AUX_REGWEN` | RW0C | 0x1 | Auxiliary Control Register configuration enable bit. If this is cleared to 0, the Auxiliary Control Register cannot be written anymore. |

### TRIGGER

*Table 326. aes.TRIGGER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `PRNG_RESEED` | W | 0x1 | Keep continuing with the current states of the internal pseudo-random number generators used for register clearing and masking (0) or perform a reseed of the internal states from the connected entropy source (1). If the KEY_TOUCH_FORCES_RESEED bit in the Auxiliary Control Register is set to one, this trigger will automatically get set after providing a new initial key. |
| 2 | `DATA_OUT_CLEAR` | W | 0x1 | Keep current values in Output Data registers (0) or clear those registers with pseudo-random data (1). |
| 1 | `KEY_IV_DATA_IN_CLEAR` | W | 0x1 | Keep current values in Initial Key, internal Full Key and Decryption Key registers, IV registers and Input Data registers (0) or clear all those registers with pseudo-random data (1). |
| 0 | `START` | W | 0x0 | Keep AES unit paused (0) or trigger the encryption/decryption of one data block (1). This trigger is cleared to `0` if MANUAL_OPERATION=0 or if MODE=AES_NONE (see Control Register). |

### STATUS

*Table 327. aes.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:7 | `Reserved` | - | - | Reserved |
| 6 | `ALERT_FATAL_FAULT` | R | 0x0 | No fatal fault has occurred inside the AES unit (0). A fatal fault has occurred and the AES unit needs to be reset (1). Examples for fatal faults include i) storage errors in the Control Register, ii) if any internal FSM enters an invalid state, iii) if any sparsely encoded signal takes on an invalid value, iv) errors in the internal round counter, v) escalations triggered by the life cycle controller, and vi) fatal integrity failures on the TL-UL bus. |
| 5 | `ALERT_RECOV_CTRL_UPDATE_ERR` | R | 0x0 | An update error has not occurred (0) or has occurred (1) in the shadowed Control Register. AES operation needs to be restarted by re-writing the Control Register. |
| 4 | `INPUT_READY` | R | 0x0 | The AES unit is ready (1) or not ready (0) to receive new data input via the DATA_IN registers. If the present values in the DATA_IN registers have not yet been loaded into the module this flag is `0` (not ready). |
| 3 | `OUTPUT_VALID` | R | 0x0 | The AES unit has no valid output (0) or has valid output data (1). |
| 2 | `OUTPUT_LOST` | R | 0x0 | All previous output data has been fully read by the processor (0) or at least one previous output data block has been lost (1). It has been overwritten by the AES unit before the processor could fully read it. Once set to `1`, this flag remains set until AES operation is restarted by re-writing the Control Register. The primary use of this flag is for design verification. This flag is not meaningful if MANUAL_OPERATION=0 (see Control Register). |
| 1 | `STALL` | R | 0x0 | The AES unit is not stalled (0) or stalled (1) because there is previous output data that must be read by the processor before the AES unit can overwrite this data. This flag is not meaningful if MANUAL_OPERATION=1 (see Control Register). |
| 0 | `IDLE` | R | 0x0 | The AES unit is idle (1) or busy (0). This flag is `0` if one of the following operations is currently running: i) encryption/decryption, ii) register clearing or iii) PRNG reseeding. This flag is also `0` if an encryption/decryption is running but the AES unit is stalled. |

### CTRL_GCM_SHADOWED

*Table 328. aes.CTRL_GCM_SHADOWED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10:6 | `NUM_VALID_BYTES` | RW | 0x10 | Number of valid bytes of the current input block. Only the last block in the GCM_AAD and GCM_TEXT phases are expected to have not all bytes marked as valid. For all other blocks, the number of valid bytes should be set to 16. Invalid input values, i.e., the value 5'b0_0000, and all other values different from 5'b1_0000 in case GCM is not supported (because disabled at compile time) are mapped to 5'b1_0000. |
| 5:0 | `PHASE` | RW | 0x1 | 6-bit one-hot field to select the phase of the Galois/Counter Mode (GCM) of operation. Invalid input values, i.e., values with multiple bits set and value 6'b00_0000, are mapped to GCM_INIT (6'b00_0001). In case support for GCM has been disabled at compile time, this field is not writable and always reads as GCM_INIT (6'b00_0001). |

### HMAC-SHA2

The HMAC component is based on the OpenTitan HMAC design.

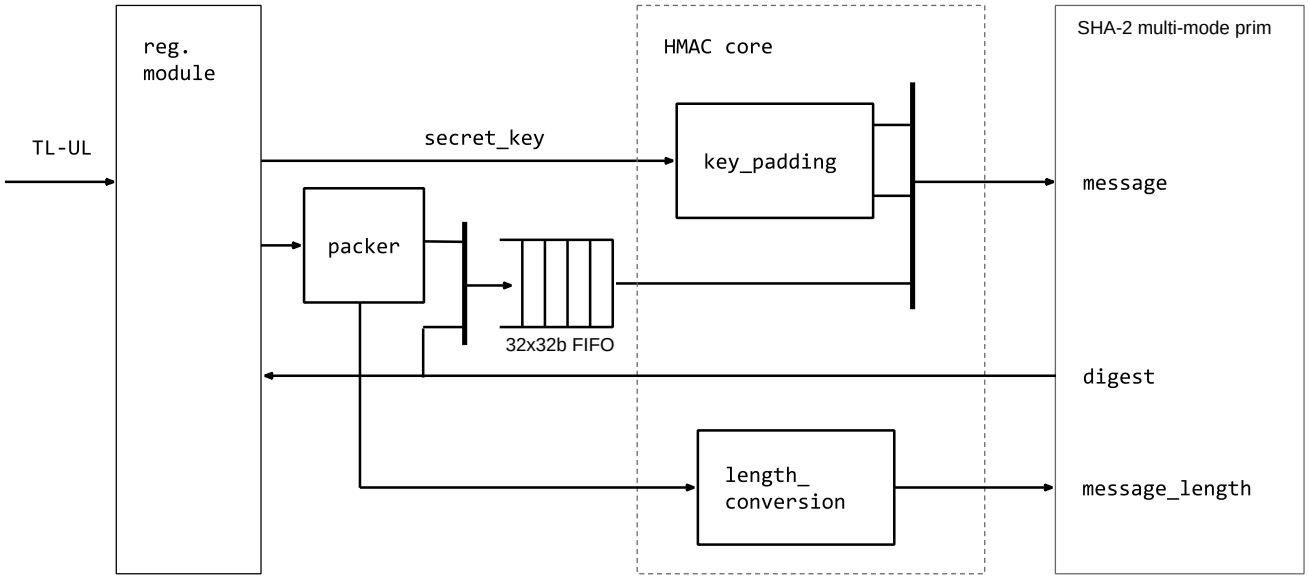


*Figure 38. OpenTitan HMAC core reference architecture*

Figure 38 shows the upstream core. The OCAH bus, key-delivery, and entropy connections are described in the integration details below.

#### OCAH Modifications

- AXI4-lite key bus slave interface replaces side loading interface
- AXI4-lite control/status bus slave interface

The crypto interconnect narrows HMAC's host path from 64-bit AXI to 32-bit AXI, converts and isolates it as AXI-Lite, then `hmac_wrapper` bridges it to TL-UL for the `tt_hmac` core. The address path subtracts the HMAC base address and masks to a 13-bit window (8 KB), accommodating both the register space and the message FIFO window at offset `0x1000`. HMAC exposes a single fatal alert channel (index `[0]` in OpenTitan Alert Termination), three interrupts (`done`, `fifo_empty`, `err`), and an `idle_o` status output. Unlike the other crypto wrappers, HMAC has no EDN connection.

The Key Manager delivers a 256-bit HMAC key as eight words per share, in the format given in Key Manager Key Delivery. HMAC has no select bit: the delivered key is used in preference to the engine's software key registers whenever `KEY_CTRL.KEY_VALID` is set.

The delivered key is most-significant-bit justified in the engine's 1024-bit key field, and the engine takes its key length from `CFG.key_length`, which the delivery path does not write; only the 256-bit encoding matches the delivered key. An unset or unsupported length raises a configuration error when hashing starts. A supported length that disagrees with the delivered key does not raise an error: the engine takes key material from the top of the 1024-bit field, so the padding below the delivered key is mixed into the key and the resulting digest is wrong but repeatable.

Consult the OpenTitan HMAC documentation for the authoritative core functionality of the IP. The Programmer's Guide gives the programming guidance for this block. The register map follows below.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each

register and its fields.

### Detailed Register Map

### HMAC Accelerator

Accelerator for SHA-2 256/384/512-based keyed HMAC and the hash function

**NOTE**

**Register Arrays:**

- **KEY[32]**: 32 registers, base 0x24, stride 0x4
- **DIGEST[16]**: 16 registers, base 0xA4, stride 0x4

*Table 329. HMAC Accelerator register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | CFG | RW | - |
| 0x14 | CMD | W | - |
| 0x18 | STATUS | R | - |
| 0x1C | ERR_CODE | R | - |
| 0x20 | WIPE_SECRET | W | - |
| 0x24 - 0xA0 | KEY[32] | W | - |
| 0xA4 - 0xE0 | DIGEST[16] | RW | - |
| 0xE4 | MSG_LENGTH_LOWER | RW | - |
| 0xE8 | MSG_LENGTH_UPPER | RW | - |

### INTR_STATE

*Table 330. hmac.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `hmac_err` | RW1C | 0x0 | HMAC error has occurred. ERR_CODE register shows which error occurred. |
| 1 | `fifo_empty` | R | 0x0 | The message FIFO is empty. This interrupt is raised only if the message FIFO is actually writable by software, i.e., if all of the following conditions are met: i) The HMAC block is not running in HMAC mode and performing the second round of computing the final hash of the outer key as well as the result of the first round using the inner key. ii) Software has not yet written the Process or Stop command to finish the hashing operation. For the interrupt to be raised, the message FIFO must also have been full previously. Otherwise, the hardware empties the FIFO faster than software can fill it and there is no point in interrupting the software to inform it about the message FIFO being empty. |
| 0 | `hmac_done` | RW1C | 0x0 | HMAC/SHA-2 has completed. |

### INTR_ENABLE

*Table 331. hmac.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `hmac_err` | RW | 0x0 | Enable interrupt when !!INTR_STATE.hmac_err is set. |
| 1 | `fifo_empty` | RW | 0x0 | Enable interrupt when !!INTR_STATE.fifo_empty is set. |
| 0 | `hmac_done` | RW | 0x0 | Enable interrupt when !!INTR_STATE.hmac_done is set. |

### INTR_TEST

*Table 332. hmac.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `hmac_err` | W | 0x0 | Write 1 to force !!INTR_STATE.hmac_err to 1. |
| 1 | `fifo_empty` | W | 0x0 | Write 1 to force !!INTR_STATE.fifo_empty to 1. |
| 0 | `hmac_done` | W | 0x0 | Write 1 to force !!INTR_STATE.hmac_done to 1. |

### ALERT_TEST

*Table 333. hmac.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `fatal_fault` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### CFG

*Table 334. hmac.CFG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | `Reserved` | - | - | Reserved |
| 14:9 | `key_length` | RW | 0x20 | Key length configuration.<br><br>This is a 6-bit one-hot encoded field to configure the key length for HMAC.<br><br>The HMAC can be programmed with the following key lengths: 128-bit, 256-bit, 384-bit, 512-bit and 1024-bit. But the HMAC supports any arbitrary key length: the software should configure the HMAC with the next largest supported key length and concatenate zeros to reach the programmed key length. The position of these zeros depends on the endianness, thus on the programmed [`CFG.key_swap`](registers.md#cfg—key_swap). For example, for an 80-bit key, HMAC should be configured with an 128-bit key length, fed with the 80-bit key and with 48 zero-bits.<br><br>Note that the key length cannot be greater than the block size: up to 1024-bit for SHA-2 384/512 and up to 512-bit for SHA-2 256. The value of this register is irrelevant when only SHA-2 (not keyed HMAC) is configured. However, for HMAC mode (`hmac_en == 1`), when HMAC is triggered to start while !!KEY_LENGTH holds `Key_None` or !!KEY_LENGTH holds `Key_1024` for !!DIGEST_SIZE = `SHA2_256`, starting is blocked and an error is signalled to SW. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 8:5 | `digest_size` | RW | 0x8 | Digest size configuration.<br><br>This is a 4-bit one-hot encoded field to select digest size for either HMAC or SHA-2. Invalid/unsupported values, i.e., values that don't correspond to SHA2_256, SHA2_384, or SHA2_512, are mapped to SHA2_None. |
| 4 | `key_swap` | RW | 0x0 | Key register byte swap (deprecated: KEY path is for DV only; production uses key manager sideload).<br><br>If 1 the endianness of each KEY_* register is swapped. Default value (value 0) is big endian representation of the KEY_* CSRs. |
| 3 | `digest_swap` | RW | 0x0 | Digest register byte swap.<br><br>If 1 the value in each digest output register is converted to big-endian byte order. This setting does not affect the order of the digest output registers, !!DIGEST_0 still contains the first 4 bytes of the digest. |
| 2 | `endian_swap` | RW | 0x0 | Endian swap.<br><br>If 0, each value will be added to the message in little-endian byte order. The value is written to MSG_FIFO same to the SW writes. If 1, then each individual multi-byte value, regardless of its alignment, written to !!MSG_FIFO will be added to the message in big-endian byte order. A message written to !!MSG_FIFO one byte at a time will not be affected by this setting. From a hardware perspective byte swaps are performed on a TL-UL word granularity. |
| 1 | `sha_en` | RW | - | SHA-2 enable.<br><br>If 0, the SHA engine will not initiate compression, this is used to stop operation of the SHA-2 engine until configuration has been done. When the SHA-2 engine is disabled the digest is cleared. |
| 0 | `hmac_en` | RW | - | HMAC datapath enable.<br><br>If this bit is 1, HMAC operates when `hash_start` toggles. |

### CMD

*Table 335. hmac.CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `hash_continue` | W1C | - | When 1 is written to this field, SHA-2 or HMAC will continue hashing based on the current hash in the digest registers and the message length, which both have to be restored to switch context. |
| 2 | `hash_stop` | W1C | - | When 1 is written to this field, SHA-2 or HMAC will afterwards set the `hmac_done` interrupt as soon as the current block has been hashed. The hash can then be read from the registers !!DIGEST_0 to !!DIGEST_15. Together with the message length in !!MSG_LENGTH_LOWER and !!MSG_LENGTH_UPPER, this forms the information that has to be saved before switching context. |
| 1 | `hash_process` | W1C | - | If 1 is written to this field, SHA-2 or HMAC calculates the digest or signing based on currently received message. |
| 0 | `hash_start` | W1C | - | If 1 is written into this field, SHA-2 or HMAC begins its operation. CPU must configure relative information first, such as the digest size, secret key and the key length. |

### STATUS

*Table 336. hmac.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:4 | `fifo_depth` | R | - | FIFO entry count. |
| 3 | `Reserved` | - | - | Reserved |
| 2 | `fifo_full` | R | - | FIFO full. Data written to the FIFO whilst it is full will cause back-pressure on the interconnect |
| 1 | `fifo_empty` | R | 0x1 | FIFO empty |
| 0 | `hmac_idle` | R | 0x1 | HMAC idle status. When IDLE, the `DIGEST` and the `MSG_LENGTH_LOWER` /`MSG_LENGTH_UPPER` can be written to from SW which enables restoring context (to support context switching). |

### ERR_CODE

*Table 337. hmac.ERR_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `err_code` | R | 0x0 | If an error interrupt occurs, this register has information of error cause. Please take a look at `hw/ip/prim/rtl/prim_sha2_pkg.sv:err_code_e enum type. |

### WIPE_SECRET

*Table 338. hmac.WIPE_SECRET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `secret` | W | - | Secret value |

### KEY[32]

*Table 339. hmac.KEY[32] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key_0` | W | - | 32-bit chunk of 1024-bit secret key |

### DIGEST[16]

*Table 340. hmac.DIGEST[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `digest_0` | RW | - | 32-bit chunk of 512-bit digest |

### MSG_LENGTH_LOWER

*Table 341. hmac.MSG_LENGTH_LOWER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `v` | RW | - | Message Length [31:0] |

#### MSG_LENGTH_UPPER

*Table 342. hmac.MSG_LENGTH_UPPER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | v | RW | - | Message Length [63:32] |

### KMAC-SHA3

The module is based on the OpenTitan KMAC design.

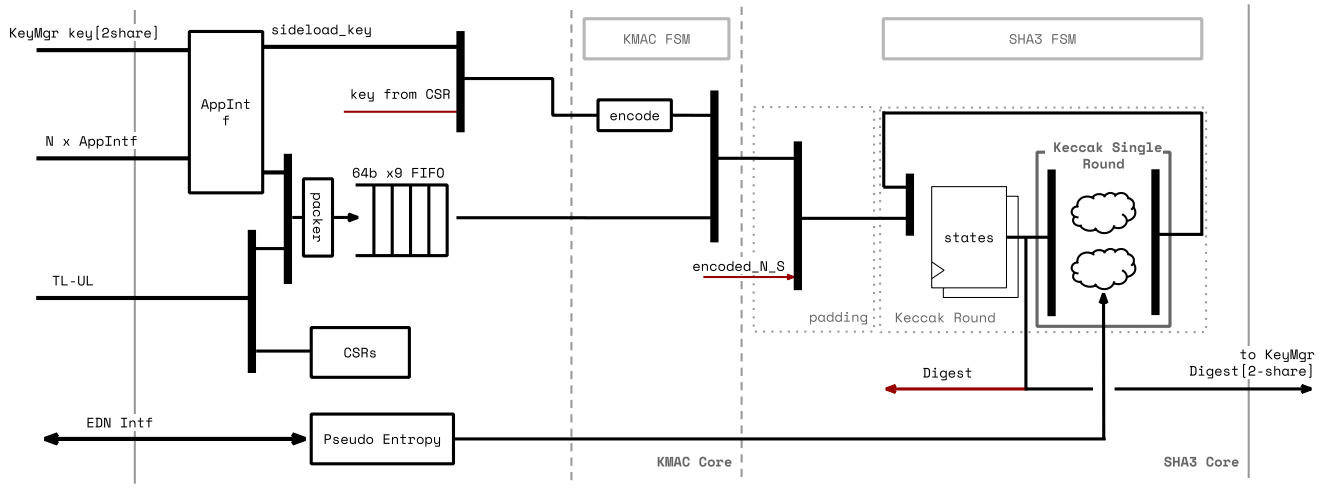


*Figure 39. OpenTitan KMAC core reference architecture*

Figure 39 shows the upstream core. The OCAH bus, key-delivery, and entropy connections are described in the integration details below.

#### OCAH Modifications

- AXI4-lite key bus slave interface replaces side loading interface
- AXI4-lite control/status bus slave interface

The KMAC host path uses the same conversion and isolation sequence as AES. The address path subtracts the KMAC base address and masks to a 12-bit window (4 KB) before the `axi_to_axi_lite` stage, aligning the internal decoder with the OpenTitan register block. `kmac_wrapper` bridges the isolated AXI-Lite path to TL-UL. EDN is handled identically to AES: KMAC is one of the four clients arbitrated by `u_axis_edn_crypto_s3c_scan` (the shared `drbg_axis_edn_adapter`), served from the same post-CSRNG DRBG AXI-Stream feed. The `app_i` application interface is tied off since the SEP does not use the KMAC hardware application path. Life-cycle escalation is inactive. KMAC occupies alert indices `[6:5]` in OpenTitan Alert Termination and exports three interrupts: `done`, `fifo_empty`, and `err`.

The Key Manager delivers a 256-bit KMAC key as eight words per share, in the format given in Key Manager Key Delivery. In KMAC, `CFG_SHADOWED.sideload` selects the delivered key; while that bit is 0 the engine uses its software key registers. Unlike AES and HMAC, software does not declare a key length for a delivered key: the engine applies the delivered key's length automatically and ignores its key-length register for that operation. Beginning a keyed operation that uses the delivered key while `KEY_CTRL.KEY_VALID` is 0 raises an error instead of producing a digest; OTBN also reports an invalid key error.

Consult the OpenTitan KMAC documentation for the authoritative core functionality of the IP. The Programmer's Guide

gives the programming guidance for this block. The register map follows below.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### KMAC Accelerator

Accelerator for Keccak-based keyed hash message authentication code and SHA-3 hash functions; with SCA and FI countermeasures

**NOTE**

**Register Arrays:**

- **KEY_SHARE0[16]**: 16 registers, base 0x30, stride 0x4
- **KEY_SHARE1[16]**: 16 registers, base 0x70, stride 0x4
- **PREFIX[11]**: 11 registers, base 0xB4, stride 0x4

*Table 343. KMAC Accelerator register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |
| 0xC | ALERT_TEST | W | - |
| 0x10 | CFG_REGWEN | R | - |
| 0x14 | CFG_SHADOWED | RW | - |
| 0x18 | CMD | W | - |
| 0x1C | STATUS | R | - |
| 0x20 | ENTROPY_PERIOD | RW | - |
| 0x24 | ENTROPY_REFRESH_HASH_CNT | R | - |
| 0x28 | ENTROPY_REFRESH_THRESHOLD_SHADOWED | RW | - |
| 0x2C | ENTROPY_SEED | W | - |
| 0x30 - 0x6C | KEY_SHARE0[16] | W | - |
| 0x70 - 0xAC | KEY_SHARE1[16] | W | - |
| 0xB0 | KEY_LEN | W | - |
| 0xB4 - 0xDC | PREFIX[11] | RW | - |
| 0xE0 | ERR_CODE | R | - |

### INTR_STATE

*Table 344. kmac.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `kmac_err` | RW1C | 0x0 | KMAC/SHA3 error occurred. ERR_CODE register shows the details |
| 1 | `fifo_empty` | R | 0x0 | The message FIFO is empty. This interrupt is raised only if the message FIFO is actually writable by software, i.e., if all of the following conditions are met: i) The KMAC block is not exercised by a hardware application interface. ii) The SHA3 block is in the Absorb state. iii) Software has not yet written the Process command to finish the absorption process. For the interrupt to be raised, the message FIFO must also have been full previously. Otherwise, the hardware empties the FIFO faster than software can fill it and there is no point in interrupting the software to inform it about the message FIFO being empty. |
| 0 | `kmac_done` | RW1C | 0x0 | KMAC/SHA3 absorbing has been completed |

### INTR_ENABLE

*Table 345. kmac.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `kmac_err` | RW | 0x0 | Enable interrupt when !!INTR_STATE.kmac_err is set. |
| 1 | `fifo_empty` | RW | 0x0 | Enable interrupt when !!INTR_STATE.fifo_empty is set. |
| 0 | `kmac_done` | RW | 0x0 | Enable interrupt when !!INTR_STATE.kmac_done is set. |

### INTR_TEST

*Table 346. kmac.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `kmac_err` | W | 0x0 | Write 1 to force !!INTR_STATE.kmac_err to 1. |
| 1 | `fifo_empty` | W | 0x0 | Write 1 to force !!INTR_STATE.fifo_empty to 1. |
| 0 | `kmac_done` | W | 0x0 | Write 1 to force !!INTR_STATE.kmac_done to 1. |

### ALERT_TEST

*Table 347. kmac.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `fatal_fault_err` | W | 0x0 | Write 1 to trigger one alert event of this kind. |
| 0 | `recov_operation_err` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### CFG_REGWEN

*Table 348. kmac.CFG_REGWEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `en` | R | 0x1 | Configuration enable. |

### CFG_SHADOWED

*Table 349. kmac.CFG_SHADOWED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:27 | `Reserved` | - | - | Reserved |
| 26 | `en_unsupported_modestrength` | RW | 0x0 | Enable Unsupported Mode and Strength configs.<br><br>SW may set this field for KMAC to move forward with unsupported Keccak Mode and Strength configurations, such as cSHAKE512.<br><br>If not set, KMAC won't propagate the SW command (CmdStart) to the rest of the blocks (AppIntf, KMAC Core, SHA3). |
| 25 | `Reserved` | - | - | Reserved |
| 24 | `entropy_ready` | RW | 0x0 | Entropy Ready status.<br><br>Software sets this field to allow the entropy generator in KMAC to fetch the entropy and run. |
| 23:21 | `Reserved` | - | - | Reserved |
| 20 | `msg_mask` | RW | 0x0 | Message Masking with PRNG.<br><br>If 1, KMAC applies PRNG to the input messages to the Keccak module when KMAC mode is on. |
| 19 | `entropy_fast_process` | RW | 0x0 | Entropy Fast process mode.<br><br>If 1, entropy logic uses garbage data while not processing the KMAC key block. It will re-use previous entropy value and will not expand the entropy when it is consumed. Only it refreshes the entropy while processing the secret key block. This process should not be used if SCA resistance is required because it may cause side channel leakage. |
| 18 | `Reserved` | - | - | Reserved |
| 17:16 | `entropy_mode` | RW | 0x0 | Entropy Mode<br><br>Using this field, software can configure mode of operation of the internal pseudo-random number generator (PRNG).<br><br>Note that once software has configured a value different from `idle_mode` after reset, and set the !!CFG_SHADOWED.entropy_ready bit afterwards, the PRNG configuration cannot be changed anymore (unless an EDN request timeout occurs when running in EDN mode). Further writes to this field will change the value received upon reading the register, but they will be ignored by the internal hardware. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12 | `sideload` | RW | 0x1 | Sideloaded Key. Deprecated: defaults to 1 (sideload). Set to 0 for DV testing with SW key path only.<br><br>If 1, KMAC uses KeyMgr sideloaded key for SW initiated KMAC operation. KMAC uses the sideloaded key regardless of this configuration when KeyMgr initiates the KMAC operation for Key Derivation Function (KDF). |
| 11:10 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 9 | `state_endianness` | RW | 0x0 | State Endianness.<br><br>If 1 then each individual word in the !!STATE output register is converted to big-endian byte order. The order of the words in relation to one another is not changed. This setting does not affect how the state is interpreted during computation. |
| 8 | `msg_endianness` | RW | 0x0 | Message Endianness.<br><br>If 1 then each individual multi-byte value, regardless of its alignment, written to !!MSG_FIFO will be added to the message in big-endian byte order. If 0, each value will be added to the message in little-endian byte order. A message written to !!MSG_FIFO one byte at a time will not be affected by this setting. From a hardware perspective byte swaps are performed on a TL-UL word granularity. |
| 7:6 | `Reserved` | - | - | Reserved |
| 5:4 | `mode` | RW | 0x0 | Keccak hashing mode.<br><br>This module supports SHA3 main hashing algorithm and the part of its derived functions, SHAKE and cSHAKE with limitations. This field is to select the mode. |
| 3:1 | `kstrength` | RW | 0x0 | Hashing Strength<br><br>3 bit field to select the security strength of SHA3 hashing engine. If mode field is set to SHAKE or cSHAKE, only 128 and 256 strength can be selected. Other value will result error when hashing starts. |
| 0 | `kmac_en` | RW | 0x0 | KMAC datapath enable.<br><br>If this bit is 1, the KMAC mode is active and the secret key is prepended to the incoming message. |

### CMD

*Table 350. kmac.CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10 | `err_processed` | W1C | - | When error occurs and one of the state machine stays at Error handling state, SW may process the error based on ERR_CODE, then let FSM back to the reset state |
| 9 | `hash_cnt_clr` | W1C | - | Software can set this bit to 1 to manually clear !!ENTROPY_REFRESH_HASH_CNT. |
| 8 | `entropy_req` | W1C | - | Software can set this bit to 1 to manually trigger the reseeding of the internal PRNG when running in EDN mode. This will also clear !!ENTROPY_REFRESH_HASH_CNT.<br><br>Note that the hardware may miss the trigger pulse if the module is not idle, or if the module is currently performing a reseed operation. |
| 7:6 | `Reserved` | - | - | Reserved |
| 5:0 | `cmd` | W1C | - | Issue a command to the KMAC/SHA3 IP. The command is sparse encoded. To prevent sw from writing multiple commands at once, the field is defined as enum. |

### STATUS

*Table 351. kmac.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:18 | `Reserved` | - | - | Reserved |
| 17 | `ALERT_RECOV_CTRL_UPDATE_ERR` | R | 0x0 | An update error has not occurred (0) or has occurred (1) in the shadowed Control Register. KMAC operation needs to be restarted by re-writing the Control Register. |
| 16 | `ALERT_FATAL_FAULT` | R | 0x0 | No fatal fault has occurred inside the KMAC unit (0). A fatal fault has occurred and the KMAC unit needs to be reset (1), Examples for such faults include i) TL-UL bus integrity fault ii) storage errors in the shadow registers iii) errors in the message, round, or key counter iv) any internal FSM entering an invalid state v) an error in the redundant lfsr |
| 15 | `fifo_full` | R | - | Message FIFO Full indicator |
| 14 | `fifo_empty` | R | 0x1 | Message FIFO Empty indicator.<br><br>The FIFO's `Pass` parameter is set to `1'b 1`. So, by default, if the SHA engine is ready, the write data to FIFO just passes through.<br><br>In this case, `fifo_depth` remains **0**. `fifo_empty`, however, lowers the value to **0** for a cycle, then goes back to the empty state, **1**.<br><br>See the "Message FIFO" section in the spec for the reason. |
| 13 | `Reserved` | - | - | Reserved |
| 12:8 | `fifo_depth` | R | - | Count of occupied entries in the message FIFO. |
| 7:3 | `Reserved` | - | - | Reserved |
| 2 | `sha3_squeeze` | R | - | If 1, SHA3 completes sponge absorbing stage. In this stage, SW can manually run the hashing engine. |
| 1 | `sha3_absorb` | R | - | If 1, SHA3 is receiving message stream and processing it |
| 0 | `sha3_idle` | R | 0x1 | If 1, SHA3 hashing engine is in idle state. |

### ENTROPY_PERIOD

*Table 352. kmac.ENTROPY_PERIOD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `wait_timer` | RW | 0x0 | EDN request wait timer.<br><br>The entropy module in KMAC waits up to this field in the timer pulse after it sends request to EDN module. If the timer expires, the entropy module moves to an error state and notifies to the system.<br><br>If there is a pending EDN request during wait timer update, then this update is delayed until the EDN request is complete.<br><br>If 0, the entropy module waits the EDN response always. If EDN does not respond in this configuration, the software shall reset the IP. |
| 15:10 | `Reserved` | - | - | Reserved |
| 9:0 | `prescaler` | RW | 0x0 | EDN Wait timer prescaler.<br><br>EDN Wait timer has 16 bit value. The timer value is increased when the timer pulse is generated. Timer pulse is raises when the number of the clock cycles hit this prescaler value.<br><br>The exact period of the timer pulse is unknown as the KMAC input clock may contain jitters. |

### ENTROPY_REFRESH_HASH_CNT

*Table 353. kmac.ENTROPY_REFRESH_HASH_CNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `hash_cnt` | R | 0x0 | Hash (KMAC) counter |

### ENTROPY_REFRESH_THRESHOLD_SHADOWED

*Table 354. kmac.ENTROPY_REFRESH_THRESHOLD_SHADOWED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `threshold` | RW | 0x0 | Hash Threshold |

### ENTROPY_SEED

*Table 355. kmac.ENTROPY_SEED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `seed` | W | - | 32-bit chunk of the entropy generator seed |

### KEY_SHARE0[16]

*Table 356. kmac.KEY_SHARE0[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key_0` | W | - | 32-bit chunk of up-to 512-bit Secret Key |

### KEY_SHARE1[16]

*Table 357. kmac.KEY_SHARE1[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key_0` | W | - | 32-bit chunk of up-to 512-bit Secret Key |

### KEY_LEN

*Table 358. kmac.KEY_LEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `len` | W | 0x0 | Key length choice |

### PREFIX[11]

*Table 359. kmac.PREFIX[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `prefix_0` | RW | 0x0 | 32-bit chunk of Encoded NS Prefix |

**ERR_CODE**

*Table 360. kmac.ERR_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `err_code` | R | 0x0 | If the `kmac_err` interrupt occurs, this register has information on the error cause. Bits 31:24 contain the error code (please refer to `err_code_e` in `hw/ip/kmac/rtl/kmac_pkg.sv`) for the encoding, and bits 23:0 contain additional debug information. This register does **not** get cleared when the `kmac_err` interrupt state gets cleared. |

### Adams Bridge (ABR)

The module is a post-quantum accelerator based on the Caliptra Adams Bridge design.

Algorithms:

- ML-DSA-87 (FIPS 204) — key generation, signature generation and verification
- ML-KEM-1024 (FIPS 203) — key generation, encapsulation and decapsulation

ML-DSA is the post-quantum signature path intended for secure boot: each owner may be provisioned with an ML-DSA root public key alongside its classical key, with `REQUIRED_SIGNERS` selecting which owners must sign and `REQUIRED_ALGS` selecting, per owner, which of the two algorithms it must sign with (see [life-cycle-controller-lcc-life-cycle-controller-lcc]). SEP ROM does not use this path today — it verifies a single classical RSA-3072 signature, and combined classical + PQC signing is deferred to a later release.

The upstream ML-DSA and ML-KEM algorithm datapaths are shown in Figure 40 and Figure 41, respectively.

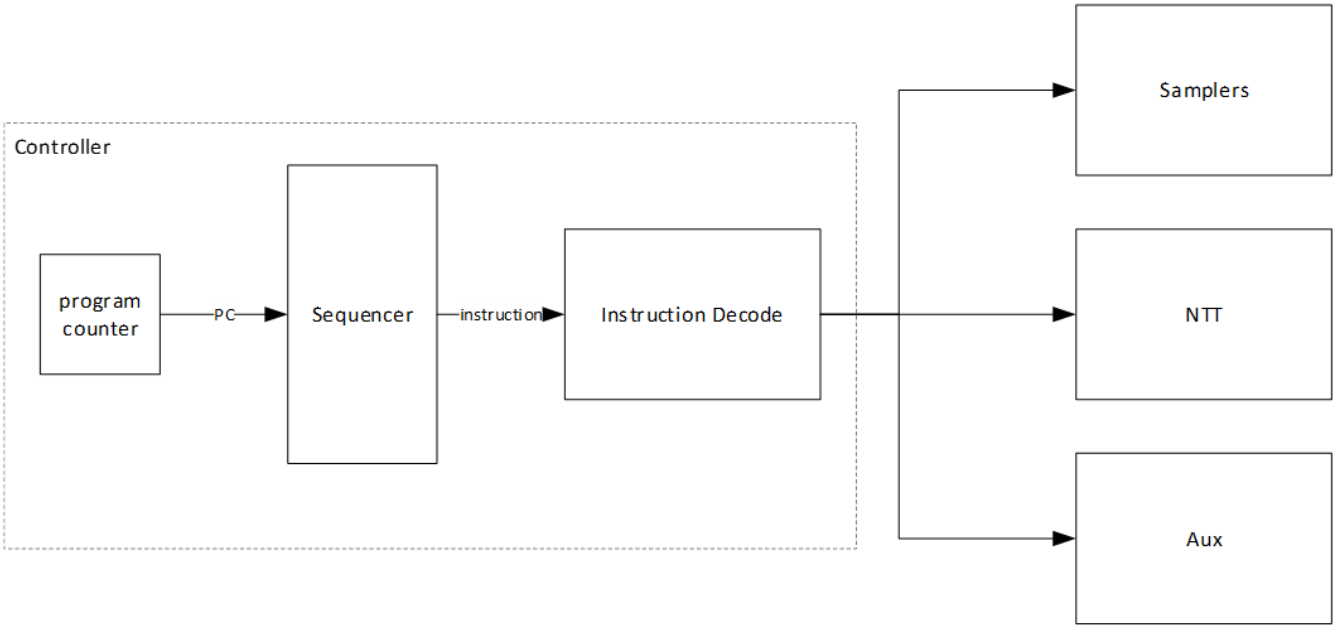


*Figure 40. Adams Bridge ML-DSA reference architecture*

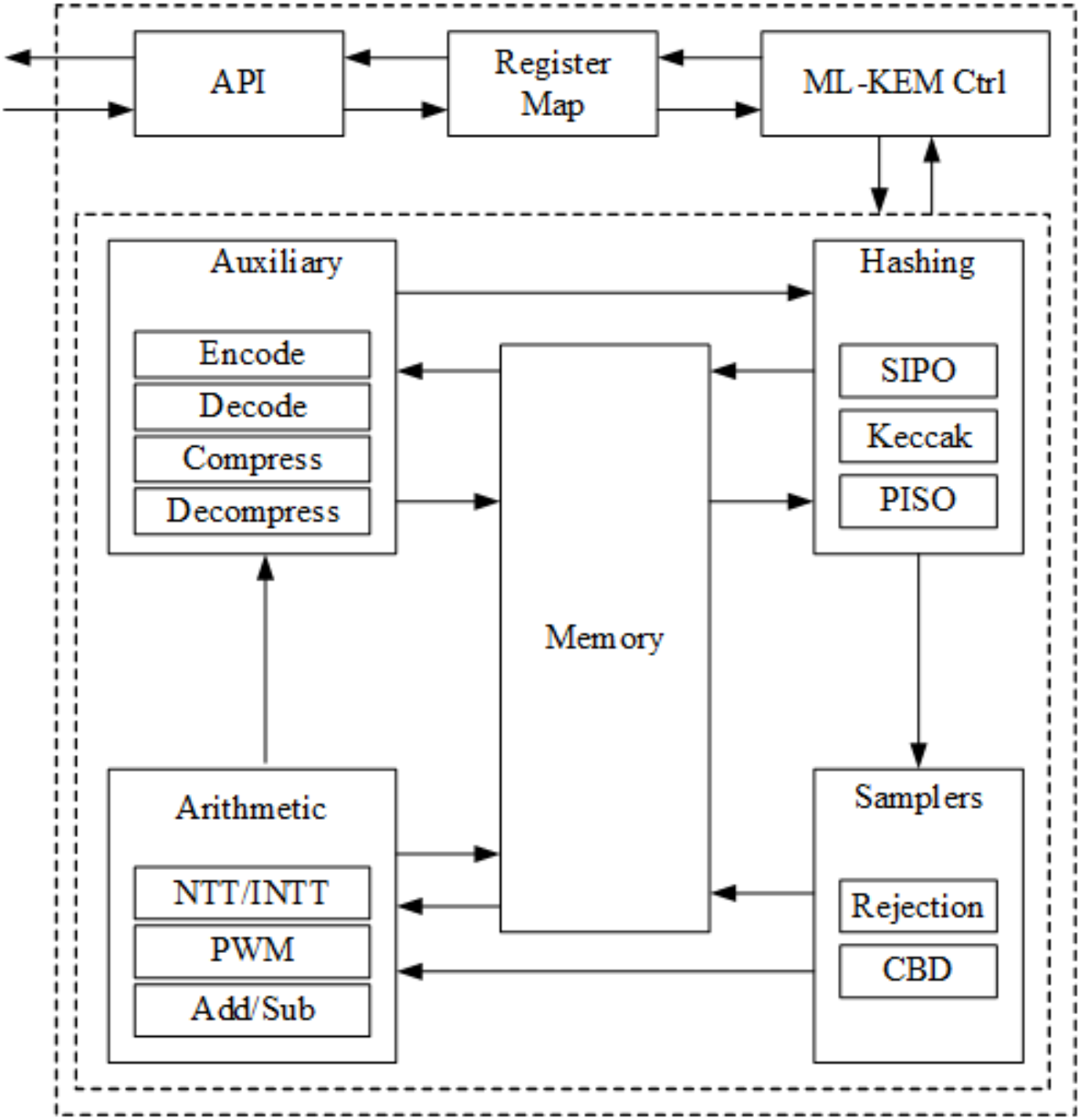


*Figure 41. Adams Bridge ML-KEM reference architecture*

These upstream diagrams describe the algorithm datapaths. The OCAH wrapper provides the AXI and key interfaces and exposes the working SRAM ports to the integrator, as described below.

**OCAH Modifications**

- AXI4-lite key bus slave interface replaces the Caliptra Key Vault
- AXI4 control/status bus slave interface, bridged to the core's native AHB-lite

ABR is the one accelerator in the subsystem that is not an OpenTitan TL-UL design, so its bus pipeline ends in AHB-lite rather than TL-UL. `sep_crypto_abr_wrapper` narrows its 64-bit AXI4 slave port to 32 bits, converts it to AXI4-lite, and

drives `abr_top` through `axi_lite_to_ahb`, which issues one single AHB transfer at a time and returns its response before it accepts the next access. The wrapper is port `SEP_CRYPTO_AXI_ABR` of the crypto demux and occupies a 64 kB aperture at `0x1094_0000`.

Every ABR register is 32 bits wide, and the aperture is accessed in words:

- A 32-bit access reads or writes one register.
- A 64-bit access at an 8-byte-aligned address is carried out as two 32-bit accesses to the two registers it covers, low address first. A 64-bit access whose address has bit 2 set reaches only the upper register.
- Each word of a write is taken by its own strobe. A word whose strobe is empty is not written, so a 64-bit write that strobes one word writes that word alone, with OKAY. A word whose strobe is partial is not written, and the access is answered with SLVERR; in a 64-bit write the other word is still written when its strobe is full. `abr_top` ties the register block's byte enables high and the AHB slave zero-extends a narrow write, so a partial write would otherwise overwrite the rest of the register.
- A narrow read is performed as a read of the whole register that contains it. Only the bytes it names are returned; the other byte lanes read as zero.

Any number of reads and writes may be outstanding on the aperture from any mix of masters. They are performed one at a time, and each returns the value of the address it names.

The crypto demux decodes the aperture only as far as the register map extends. It forwards an offset inside that extent to ABR and answers every access past it with DECERR itself, whatever its size or strobes.

Inside the extent there is no further decode. The vendored register block consumes every offset bit and applies no address-validity or permission check, so the extent is fully consumed but not fully populated: register objects sit at coarse offsets with unused space between them.

The SEP memory map states how a word access to an unpopulated offset answers, inside the extent and past it. A partial-word write inside the extent is answered with SLVERR wherever it lands (ABR access size). ABR never drives `hresp` to error, so the AHB error response its slave interface implements is unreachable. A burst is answered with DECERR by the demux (Single-Beat Access Only). A malformed ABR register address inside the extent therefore returns no bus error.

**NOTE** The absence of decode inside the extent follows from the vendored collateral rather than a SEP decision. `abr_reg.sv` is generated in the Adams Bridge repository, not by the OCAH register flow, and address-validity decode is a PeakRDL-regblock exporter option rather than a SystemRDL property; the committed file predates that option. Changing it is an upstream change.

Upstream, Adams Bridge draws its secrets from the Caliptra Key Vault. OCAH does not instantiate that vault. The Key Manager's private AXI4-Lite key bus terminates inside the wrapper at the `abr_wrapper_key_reg` CSR block — the same convention as the AES and OTBN wrappers — and `sep_abr_kv_shim` presents the recovered dual-share secrets to `abr_top` over the Caliptra `kv_read` / `kv_write` port protocol. Only the Key Vault client helpers are vendored; the vault storage block is not compiled. This leaves the vendored core unmodified while giving ABR the same kind of key path as every other accelerator, one mastered by the Key Manager.

ABR is delivered seeds rather than keys, and is the only accelerator with more than one delivery destination. Four blocks are qualified independently — `MLDSA_SEED`, `MLKEM_SEED_D`, `MLKEM_SEED_Z` and `MLKEM_MSG` — each carrying eight words per share with its own `KEY_CTRL.KEY_VALID`, in the format given in Key Manager Key Delivery. The ML-DSA seed and the ML-KEM message are read as single 256-bit values. The ML-KEM seed is read as one 512-bit value formed by concatenating `MLKEM_SEED_D` followed by `MLKEM_SEED_Z`, so that read completes only when both blocks are valid. An operation consumes a delivered seed only once its seed read is enabled.

Word order is the same as for every other accelerator: word `i` of a delivered block is register index `i` of the corresponding software register array (`MLDSA_SEED[i]`, `MLKEM_SEED_D[i]`, `MLKEM_SEED_Z[i]` or `MLKEM_MSG` word `i`), so a value delivered by the Key Manager and the same eight words written by software present the same seed or message to the engine. Index 0 holds the first four bytes of the FIPS 203 or FIPS 204 byte string, with byte 0 in bits `[7:0]`, as upstream Adams Bridge defines for its registers.

The ML-KEM shared key travels in the opposite direction, making this the only delivery destination that is also a source: the accelerator writes the shared key into `MLKEM_SHARED_KEY.KEY`, and on the final word sets that block's `KEY_CTRL.KEY_VALID` and `IRQ_STATUS.KEY_VALID`, which raises the gated shared-key interrupt consumed by the Key Manager. `KEY[i]` holds bytes `4i` to `4i+3` of the FIPS 203 shared key, in the same order as the engine's own `MLKEM_SHARED_KEY` register array.

**NOTE** This word order differs from the upstream Caliptra integration. The Caliptra Key Vault stores an entry most-significant word first, and the vendored controller reverses word index on its Key Vault read and write ports to match. `sep_abr_kv_shim` applies the inverse reversal, so Caliptra's documentation of Key Vault word order does not apply to the Key Manager path.

The working SRAM is brought out of the wrapper as packed `abr_mem_req_t` / `abr_mem_rsp_t` structs. Word-write channels are `we` / `re` / address / data; `sig_z` and `pk` also carry one-bit-per-byte `wstrobe`. `SEP_ABR_SRAM_LATENCY` is the macros' native registered-read latency (default 1). `SEP_ABR_MASKING_EN` (default 1) selects Adams Bridge 2-share DOM: thirteen 1R1W macros versus nine, plus a second NTT. That choice changes ABR internal behavior (sequencing, SCA, memory topology), not merely throughput.

ABR has no alert channel — it is the only crypto accelerator that reports faults solely by interrupt (Adams Bridge error and notification in SEP CPU Interrupt Vector Map), plus a gated ML-KEM shared-key interrupt consumed directly by the Key Manager. Whether the block is present in a build is controlled by `SEP_ABR_EN`, which always implies the vendored RTL's `CALIPTRA` define; when it is absent, both the crypto crossbar port and the Key Manager key bus are terminated with AXI4-Lite error slaves.

**NOTE** Adams Bridge does not auto-return to its RESET state when an operation finishes. The sequencer parks at the operation's end state with `STATUS.VALID` set, and `STATUS.READY` re-asserts only after a `zeroize` or a reset, so completion is visible in `VALID` and back-to-back operations need a `zeroize` between them.

Consult the Adams Bridge documentation for the authoritative core functionality of the IP. The Programmer's Guide gives the programming guidance for this block. Its CSRs are defined by the vendored Caliptra register package rather than by an OCAH PeakRDL source, so unlike the other accelerators they do not appear in Detailed Register Map.

#### OTBN PKA Public Key Accelerator

The module is based on the OpenTitan OTBN design.

Features:

- Fully programmable big integer processor
  - 256-bit data path with 32 entry register file
  - 32-bit control path with 32 entry register file
  - Complex flow control: conditional branching, loops, call/return with optimized hardware for performance
  - Dual status control bits to assist in big integer computation
  - Optimized modular arithmetic on big integers

- Access to entropy noise stream from the TRNG

The OTBN is programmed in assembly and its runtime driver library currently supports the following algorithms:

- ECDSA P256, P384
- ECDH P256, P384
- Ed25519
- X25519
- RSA 2k, 3k, 4k
  - Key generation, signature generation and signature verification
- SHA256, SHA-512

Since the OTBN is programmable, more algorithms can easily be developed at a later time. The microcode is loaded into SRAM based instruction memory allowing firmware updates to be made in the field.

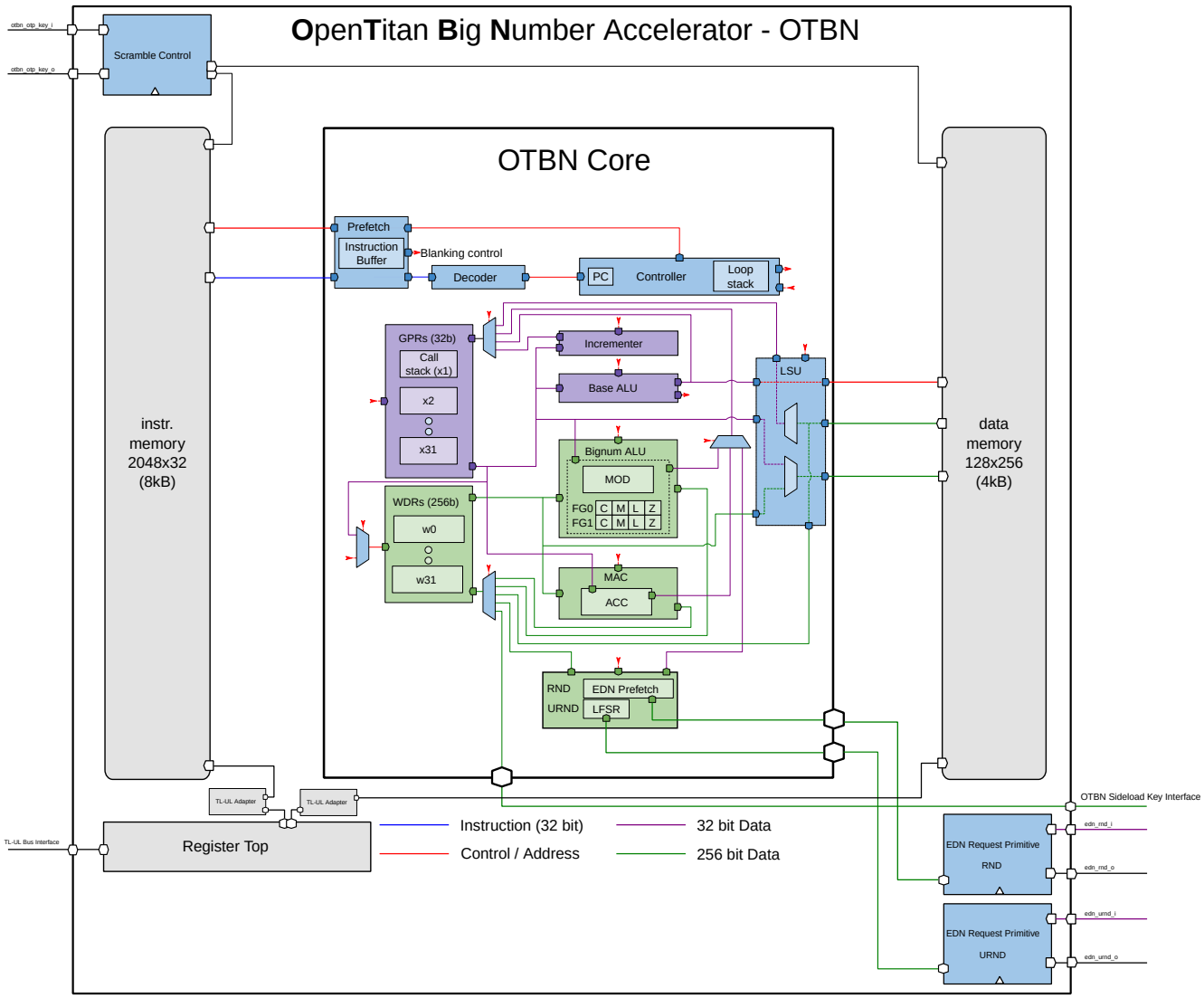


*Figure 42. OpenTitan OTBN core reference architecture*

Figure 42 shows the upstream core. The OCAH bus, key-delivery, and entropy connections are described in the

integration details below.

#### OCAH Modifications

- AXI4-lite key bus slave interface replaces side loading interface
- AXI4-lite control/status bus slave interface

The OTBN host path is the most complex OpenTitan path. The crypto interconnect applies a `CACHE_MODIFIABLE` bit fix before the `axi_dw_converter`, because the VeeR CPU may emit non-modifiable cache attributes for MMIO accesses that would be rejected by the AXI-to-AXI-Lite converter. After data-width conversion and AXI-Lite isolation, `sep_crypto_otbn_wrapper` bridges the request to TL-UL and drives the standard `otbn` core configured with `Stub = 0` and `RegFile = RegFileFF`.

OTBN's instruction and data memories are external SRAM macros; the IMEM and DMEM `prim_ram_1p_scr` ports are brought out of the wrapper to the `sep_crypto` top level. The two random-number request ports (`edn_rnd`, `edn_urnd`) are served over native-protocol EDN by the shared `drbg_axis_edn_adapter` instance `u_axis_edn_crypto_s3c_scan` in `sep_crypto`, which converts the post-CSRNG DRBG AXI-Stream feed assigned to the crypto sink domain into per-client EDN responses. The adapter forwards the per-beat FIPS tag through a generic AXI-Stream `tuser` sideband. The CSRNG seed tag and external entropy mux tag are tied high by the integration's provisional policy. A high tag prevents a FIPS-tag failure for that response but does not prove that the entropy is FIPS compliant. The OTP scramble key handshake is tied to fixed values. Life-cycle escalation and RMA request signals are tied to their inactive state. OTBN occupies alert indices `[2:1]` in OpenTitan Alert Termination and exports a single `intr_done_o` interrupt.

The Key Manager delivers a 384-bit OTBN key as twelve words per share, the widest delivery in the subsystem, in the format given in Key Manager Key Delivery. OTBN is the only accelerator that does not combine the shares in hardware. The shares are exposed to the executing program through the `KEY_S0_L` / `KEY_S0_H` and `KEY_S1_L` / `KEY_S1_H` special registers, each share split into a 256-bit low and a 128-bit high half, and the program is responsible for reconstructing the key as the XOR of the two shares. Each share is qualified independently: reading a share whose key is not valid sets `ERR_BITS.KEY_INVALID` and stops the program rather than returning key material.

Consult the OpenTitan OTBN documentation for the authoritative core functionality of the IP. The Programmer's Guide gives the programming guidance for this block. The register map follows below.

#### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### OpenTitan Big Number Accelerator

Programmable coprocessor for asymmetric cryptography with SCA and FI countermeasures

*Table 361. OpenTitan Big Number Accelerator register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | - |
| 0x4 | INTR_ENABLE | RW | - |
| 0x8 | INTR_TEST | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0xC | ALERT_TEST | W | - |
| 0x10 | CMD | W | - |
| 0x14 | CTRL | RW | - |
| 0x18 | STATUS | R | - |
| 0x1C | ERR_BITS | RW | - |
| 0x20 | FATAL_ALERT_CAUSE | R | - |
| 0x24 | INSN_CNT | RW | - |
| 0x28 | LOAD_CHECKSUM | RW | - |

### INTR_STATE

*Table 362. otbn.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `done` | RW1C | 0x0 | OTBN has completed the operation. |

### INTR_ENABLE

*Table 363. otbn.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `done` | RW | 0x0 | Enable interrupt when !!INTR_STATE.done is set. |

### INTR_TEST

*Table 364. otbn.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `done` | W | 0x0 | Write 1 to force !!INTR_STATE.done to 1. |

### ALERT_TEST

*Table 365. otbn.ALERT_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `recov` | W | 0x0 | Write 1 to trigger one alert event of this kind. |
| 0 | `fatal` | W | 0x0 | Write 1 to trigger one alert event of this kind. |

### CMD

*Table 366. otbn.CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | `cmd` | W | 0x0 | The operation to perform.<br><br>\| Value \| Name \| Description \| \|:------\|:----------\|:----------\| \| 0xd8 \| EXECUTE \| Starts the execution of the program stored in the instruction memory, starting at address zero. \| \| 0xc3 \| SEC_WIPE_DMEM \| Securely removes all contents from the data memory. \| \| 0x1e \| SEC_WIPE_IMEM \| Securely removes all contents from the instruction memory. \| |

### CTRL

*Table 367. otbn.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `software_errs_fatal` | RW | 0x0 | Controls the reaction to software errors.<br><br>When set software errors produce fatal errors, rather than recoverable errors.<br><br>Writes are ignored if OTBN is not idle. |

### STATUS

*Table 368. otbn.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `status` | R | 0x4 | Indicates the current operational state OTBN is in.<br><br>All BUSY values represent an operation started by a write to the !!CMD register.<br><br>\| Value \| Name \| Description \| \|:------\|:----------\|:----------\| \| 0x00 \| IDLE \| OTBN is idle: it is not performing any action. \| \| 0x01 \| BUSY_EXECUTE \| OTBN is busy executing software. \| \| 0x02 \| BUSY_SEC_WIPE_DMEM \| OTBN is busy securely wiping the data memory. \| \| 0x03 \| BUSY_SEC_WIPE_IMEM \| OTBN is busy securely wiping the instruction memory. \| \| 0x04 \| BUSY_SEC_WIPE_INT \| OTBN is busy securely wiping the internal state. \| \| 0xFF \| LOCKED \| OTBN is locked as reaction to a fatal error, and must be reset to unlock it again. See also the section "Reaction to Fatal Errors". \| |

### ERR_BITS

*Table 369. otbn.ERR_BITS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23 | `fatal_software` | RW | 0x0 | A `FATAL_SOFTWARE` error was observed. |
| 22 | `lifecycle_escalation` | RW | 0x0 | A `LIFECYCLE_ESCALATION` error was observed. |
| 21 | `illegal_bus_access` | RW | 0x0 | An `ILLEGAL_BUS_ACCESS` error was observed. |
| 20 | `bad_internal_state` | RW | 0x0 | A `BAD_INTERNAL_STATE` error was observed. |
| 19 | `bus_intg_violation` | RW | 0x0 | A `BUS_INTG_VIOLATION` error was observed. |
| 18 | `reg_intg_violation` | RW | 0x0 | A `REG_INTG_VIOLATION` error was observed. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 17 | dmem_intg_violation | RW | 0x0 | A DMEM_INTG_VIOLATION error was observed. |
| 16 | imem_intg_violation | RW | 0x0 | A IMEM_INTG_VIOLATION error was observed. |
| 15:9 | Reserved | - | - | Reserved |
| 8 | mai_software_error | RW | 0x0 | An MAI_SOFTWARE_ERROR error was observed. |
| 7 | rnd_fips_chk_fail | RW | 0x0 | An RND_FIPS_CHK_FAIL error was observed. |
| 6 | rnd_rep_chk_fail | RW | 0x0 | An RND_REP_CHK_FAIL error was observed. |
| 5 | key_invalid | RW | 0x0 | A KEY_INVALID error was observed. |
| 4 | loop | RW | 0x0 | A LOOP error was observed. |
| 3 | illegal_insn | RW | 0x0 | An ILLEGAL_INSN error was observed. |
| 2 | call_stack | RW | 0x0 | A CALL_STACK error was observed. |
| 1 | bad_insn_addr | RW | 0x0 | A BAD_INSN_ADDR error was observed. |
| 0 | bad_data_addr | RW | 0x0 | A BAD_DATA_ADDR error was observed. |

#### FATAL_ALERT_CAUSE

*Table 370. otbn.FATAL_ALERT_CAUSE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7 | fatal_software | R | 0x0 | A FATAL_SOFTWARE error was observed. |
| 6 | lifecycle_escalation | R | 0x0 | A LIFECYCLE_ESCALATION error was observed. |
| 5 | illegal_bus_access | R | 0x0 | A ILLEGAL_BUS_ACCESS error was observed. |
| 4 | bad_internal_state | R | 0x0 | A BAD_INTERNAL_STATE error was observed. |
| 3 | bus_intg_violation | R | 0x0 | A BUS_INTG_VIOLATION error was observed. |
| 2 | reg_intg_violation | R | 0x0 | A REG_INTG_VIOLATION error was observed. |
| 1 | dmem_intg_violation | R | 0x0 | A DMEM_INTG_VIOLATION error was observed. |
| 0 | imem_intg_violation | R | 0x0 | A IMEM_INTG_VIOLATION error was observed. |

#### INSN_CNT

*Table 371. otbn.INSN_CNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | insn_cnt | RW | 0x0 | The number of executed instructions. |

#### LOAD_CHECKSUM

*Table 372. otbn.LOAD_CHECKSUM fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | checksum | RW | 0x0 | Checksum accumulator |

### Key Manager (KM)

The Key Manager provides isolated key storage, random key generation, explicit revocation, and delivery to the cryptographic accelerators. The following sections describe its hardware architecture and firmware interface.

The immutable ROM firmware first initializes the Key Manager, then remains active to process SEP mailbox commands and service interrupts. It is both startup firmware and the resident runtime; "boot ROM" names the memory, not its only function. On an explicit firmware-load/execute command, ROM can hand control to mutable firmware in SRAM. The Key Manager Application ROM appendix of the Programmer's Guide defines the implemented services, command protocol and handover rules.

**Architecture**

The Key Manager (KM) uses a PicoRV32 CPU, private memories and a Key and Policy Vault (KPV) to separate key management from SEP host execution. The host submits commands and supplied key material through a mailbox; KM firmware validates requests, accesses the vault and writes accelerator sideload registers. The host has no direct bus access to KM key storage.

The ROM is the resident command processor as well as the startup firmware. Its mailbox commands report the hardware, ROM and SRAM firmware versions and the recoverable-error status, generate keys from the DRBG, load SEP-supplied keys, revoke keys, transfer keys to the accelerators, shred accelerator sideload keys, capture the Adams Bridge ML-KEM shared key into the vault, set cold-domain OTP read-locks, and either load and execute mutable firmware in SRAM or keep the ROM running (the command summary in the Programmer's Guide).

A single KM-mastered AXI4-Lite crossbar connects KMCSR, KPV, the DRBG sampler and mailbox to the private accelerator key interfaces and the SEP OTP/eFuse access path. The accelerator interfaces serve AES, HMAC, KMAC, OTBN and Adams Bridge. Address decoding separates their windows; command and key policy checks belong to KM firmware.

The Memory Map and Register Reference at the end of the chapter defines the address windows and all generated fields. The Key Manager Application ROM appendix of the Programmer's Guide defines commands, validation and responses. ROM/SRAM parity detects errors but does not correct them; the ROM fault handler can report an unrecoverable fault and halt the CPU.

Figure 43 shows the Key Manager CPU, vault, mailbox, and accelerator interfaces.

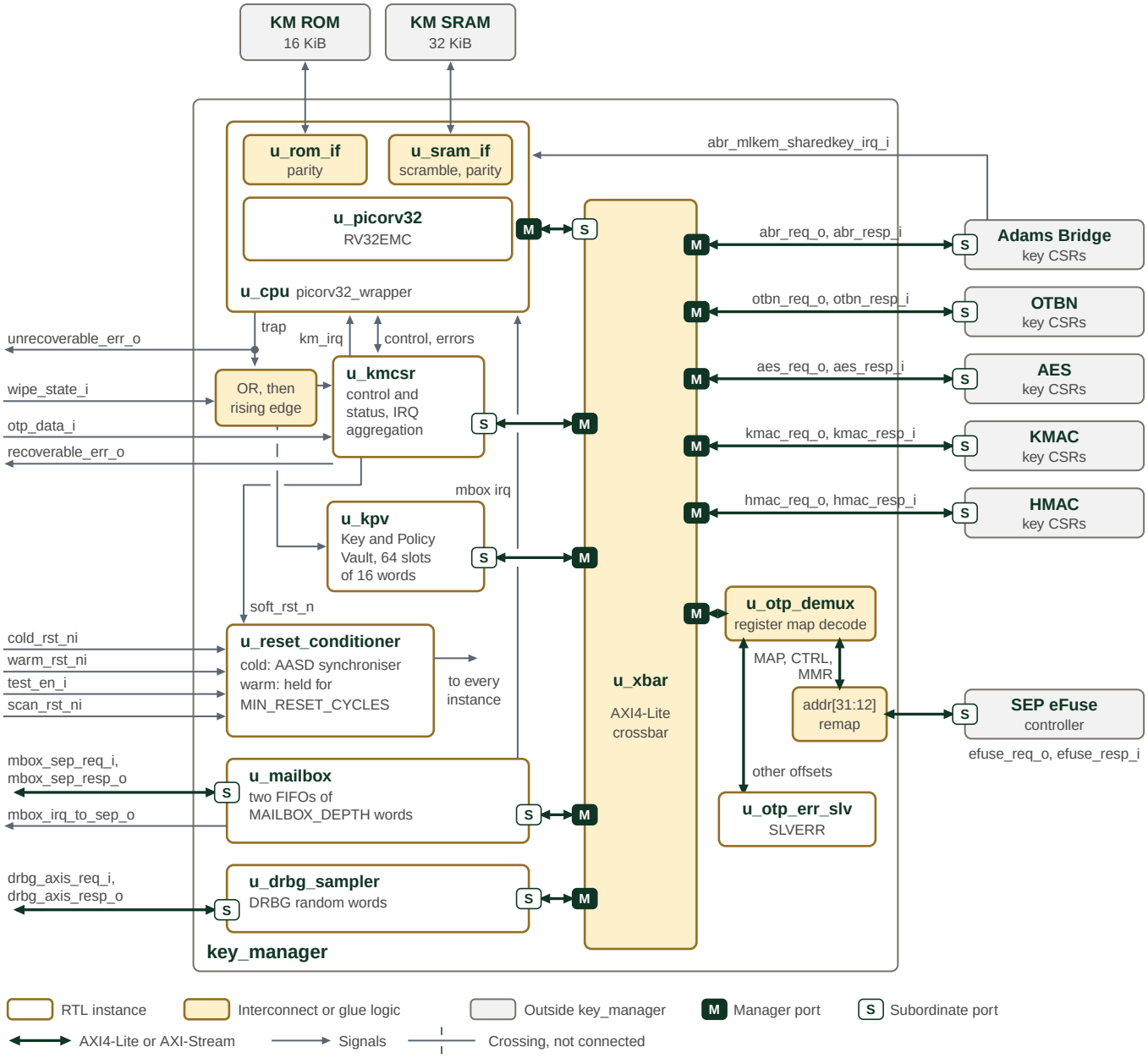


*Figure 43. Key Manager architecture*

White boxes are `key_manager` instances, cream boxes are interconnect or glue logic, and grey boxes lie outside `key_manager`. M and S mark the manager and subordinate side of each AXI4-Lite or AXI-Stream connection.

The CPU reaches the ROM and SRAM through direct memory ports rather than the crossbar. The ROM and SRAM hard macros are instantiated at integration level, so `key_manager` exposes `rom_mem_*` and `sram_mem_*` ports to them. Every other CPU access leaves the PicoRV32 AXI4-Lite master and is decoded by `u_xbar`.

The five engine ports reach only the key CSRs of each engine wrapper. In SEP each passes through its own isolation stage and carries no host traffic; the SEP host reaches the engines' data registers through the crypto interconnect, a path the KM does not see. For offsets in the MAP, CTRL and MMR register maps, the OTP/eFuse port replaces address bits `[31:12]` with `OTP_EFUSE_REMAP_BASE[31:12]` before driving `efuse_req_o`; it answers every other offset in the window with SLVERR.

A rising edge on `wipe_state_i`, or a CPU trap, zeroes the KPV and sets `IRQ_STATUS.WIPE_STATE` in KMCSR. The CPU trap also

drives `unrecoverable_err_o`. `u_kmcsr` drives `soft_rst_n` into `u_reset_conditioner` (Figure 44). `test_en_i` selects the scan reset in `u_reset_conditioner` and drives the test inputs of `u_xbar` and `u_mailbox`.

### Parameters

The Key Manager is configured through its ROM and SRAM sizes, the mailbox FIFO depth, the read-data behaviour of its memories, and the address of the eFuse controller. The main KM SRAM is 32 KB (8192 × 32-bit words), implemented with the scrambler macro `scrambler_8192x32`. The SRAM macro must return read data (`rvalid`) at least one cycle after the cycle in which it grants the read, because the SRAM interface descrambles and parity-checks each response with the address and byte strobes it captures at the grant.

The parameters of `key_manager` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Reset Conditioning

Figure 44 shows how cold and warm reset requests are conditioned for the Key Manager.

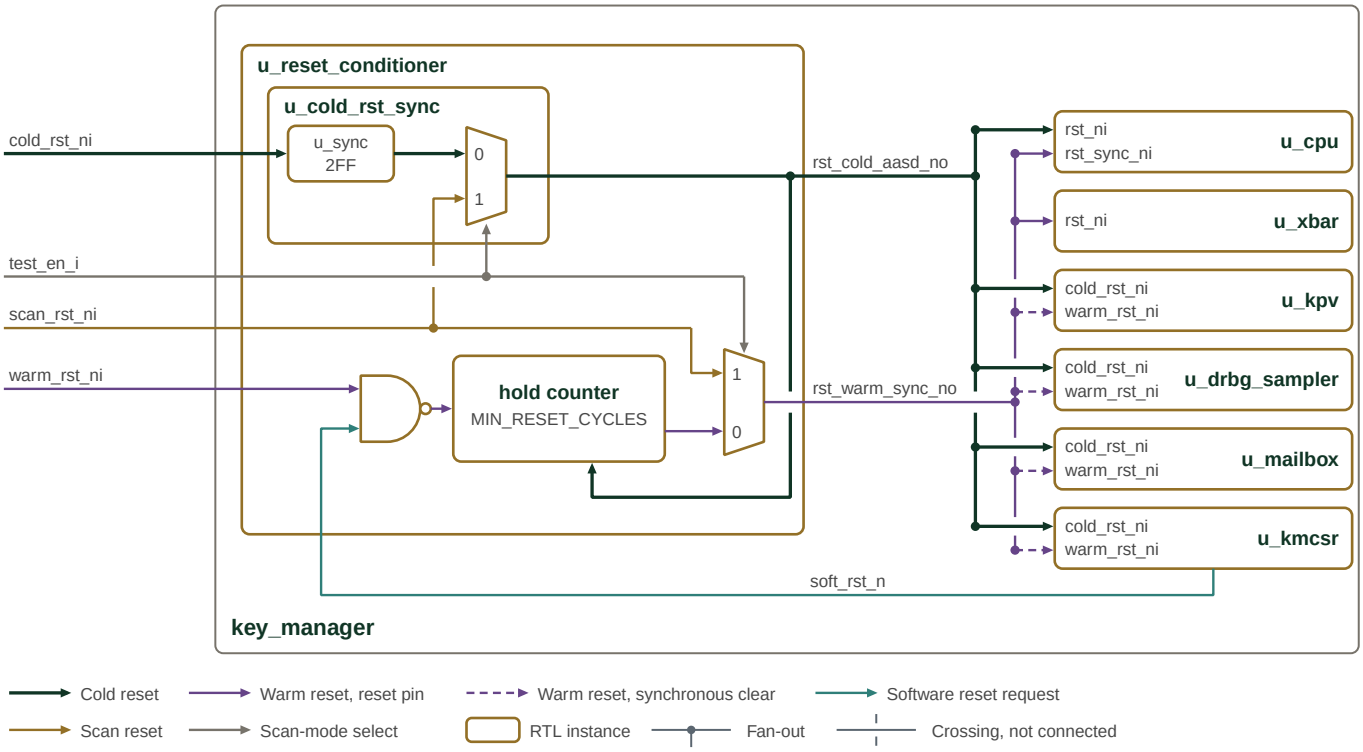


*Figure 44. Key Manager cold and warm reset conditioning*

Each reset net keeps its own colour. The cold and warm outputs of `u_reset_conditioner` run in two spines beside the instances, with the port name at each instance edge. A solid warm line drives a reset pin; a dashed warm line is a synchronous clear, which the instance samples on `clk_i` rather than using as a reset. Junction dots mark fan-out, and a gap in a line marks a crossing without connection. In test mode, `test_en_i` selects `scan_rst_ni` in both scan multiplexers.

Cold and warm resets have different scopes. Warm reset restarts the CPU and its control interfaces while retaining cold-domain state and memory contents. Retention across warm reset does not make SRAM or registers non-volatile, and reset must not be treated as a key-data wipe.

Table 373 identifies the two external reset inputs and their SEP connections.

*Table 373. Key Manager reset inputs*

| Input port | Polarity | Synchronicity | Integration tie-off (SEP) |
|---|---|---|---|
| `cold_rst_ni` | Active-low | Asynchronous | SEP system cold reset (`sep_reset_n`). |
| `warm_rst_ni` | Active-low | Synchronous to `clk_i` | SEP KM sub-component software-reset CSR bit ( `km_sw_rst_ni`). |

SEP isolates the Key Manager's AXI paths before asserting the warm reset; see Resetting the Key Manager.

`km_reset_conditioner` produces two active-low outputs:

- `rst_cold_aasd_no`: `cold_rst_ni` passes through `prim_rst_sync` for asynchronous assertion and synchronous release. There is no cold-path hold counter. Cold reset also asserts the warm path.
- `rst_warm_sync_no`: either synchronous input, `warm_rst_ni` or KMCSR `soft_rst_ni`, triggers a hold counter. Reset remains asserted for at least `MIN_RESET_CYCLES` clocks (default 10) after the trigger releases. The cold output asynchronously resets this counter; cold reset is not sampled as a synchronous trigger.

`scanmode_i` selects `scan_rst_ni` to override both paths for DFT. Table 374 identifies resettable state; it does not imply that key-data flops or memory arrays have reset values.

*Table 374. Key Manager reset-domain assignment*

| Reset domain | Components |
|---|---|
| Cold only (`rst_cold_aasd_no`) | ROM/SRAM memory interfaces and parity logic; SRAM scrambler lock state (`SCRAMBLER_CTRL.lock`, `SCRAMBLER_KEY`); the SEP-facing mailbox FIFOs and state machines; shared mailbox separator/status bits; the `BOOT_STATUS.COLD_BOOT_DONE` flag; the `OTP_READ_LOCK_COLD` register; KPV scrambler enable/lock state. |
| Warm + cold (`rst_warm_sync_no`) | PicoRV32 CPU core; the KM-internal AXI4-Lite crossbar/fabric transaction state; the KMCSR warm subset (`IRQ_STATUS`/`IRQ_ENABLE`/`IRQ_SET`, `SOFT_RST_CODE`, `RECOVERABLE_ERR`, `SRAM_LOCK` write-lock bits, `SRAM_LOCK` violation bits, `SRAM_EXEC_MODE`, the ROM lockout latch, `IRQ_ENTRY_ADDR`/`IRQ_ENTRY_LOCK`, `OTP_READ_LOCK`, `OTP_CHANGE_STATUS`, and the firmware test-bench registers); DRBG sampler configuration/status controls and request state (datapath storage has no reset value); the KPV per-slot lock bits (`lock_write`, `lock_use`, `seal`); and the KM-side mailbox write/read state machines, IRQ, control, and status fields. |

The internal AXI4-Lite fabric and generated register CPU interfaces reset on warm reset, discarding their in-flight transaction state. This is an internal reset boundary, not a completion or cancellation guarantee for an external accelerator transaction. Mailbox FIFO contents survive warm reset; ROM flushes the mailbox during boot before accepting new commands.

Writing `0x53525354` (ASCII `"SRST"`) to KMCSR `SOFT_RST_CODE` triggers the warm path. `BOOT_STATUS.COLD_BOOT_DONE` survives it: ROM sets this flag after cold-boot initialization and reads it on subsequent boots to distinguish warm restart from cold initialization.

### Mailbox Transport

The mailbox is a raw 32-bit word transport with two FIFOs: inbound SEP-to-KM and outbound KM-to-SEP, each at least 16 words deep. Hardware does not parse commands, validate CRCs or enforce key policy. Firmware defines the framing in Table 375 and Table 376; use the firmware reference for command legality, sequencing and error responses.

*Table 375. Key Manager message with payload*

| header_crc8[31:24] | message_length[23:16] | function_id[15:8] | sequence_number[7:0] |
|---|---|---|---|
| Data_1[31:0] | | | |
| Data_2[31:0] | | | |
| ... | | | |
| Data_N[31:0] | | | |

| header_crc8[31:24] | message_length[23:16] | function_id[15:8] | sequence_number[7:0] |
|---|---|---|---|
| CRC32[31:0] - checksum | | | |

Command/response messages without a data payload (message_length = 0x00) drop the payload CRC32 and only contain the header (Table 376).

*Table 376. Key Manager header-only message*

| header_crc8[31:24] | 0x00 | function_id[15:8] | sequence_number[7:0] |
|---|---|---|---|

Each FIFO word carries a separator bit that the sender sets on the last word of a message. The receiver uses it with the header length to check framing. Reads pop words; writes push words. Status reports fill level and empty/full conditions.

A write to a full FIFO drops the incoming word without changing queued data. An empty read has no data to consume. Both return SLVERR by default, with configuration allowing OKAY, and set the corresponding sticky overflow or underflow status. Each side has write-one-to-trigger flush control: either side flushes **both** FIFOs and sets the peer's flushed-by-other status.

Each side independently enables its interrupt sources. Receive-data-available and transmit-space-available are level conditions; overflow, underflow and peer-flush are sticky conditions. Their enabled OR drives one interrupt to each side: directly to the KM CPU for the KM side, and to the SEP interrupt controller for the SEP side. KM firmware validates received words before executing a command and uses the outbound FIFO for responses and fault notifications. Error handling and resynchronization are firmware operations, not hardware command filtering.

Figure 45 shows the mailbox paths used for SEP commands and Key Manager responses.

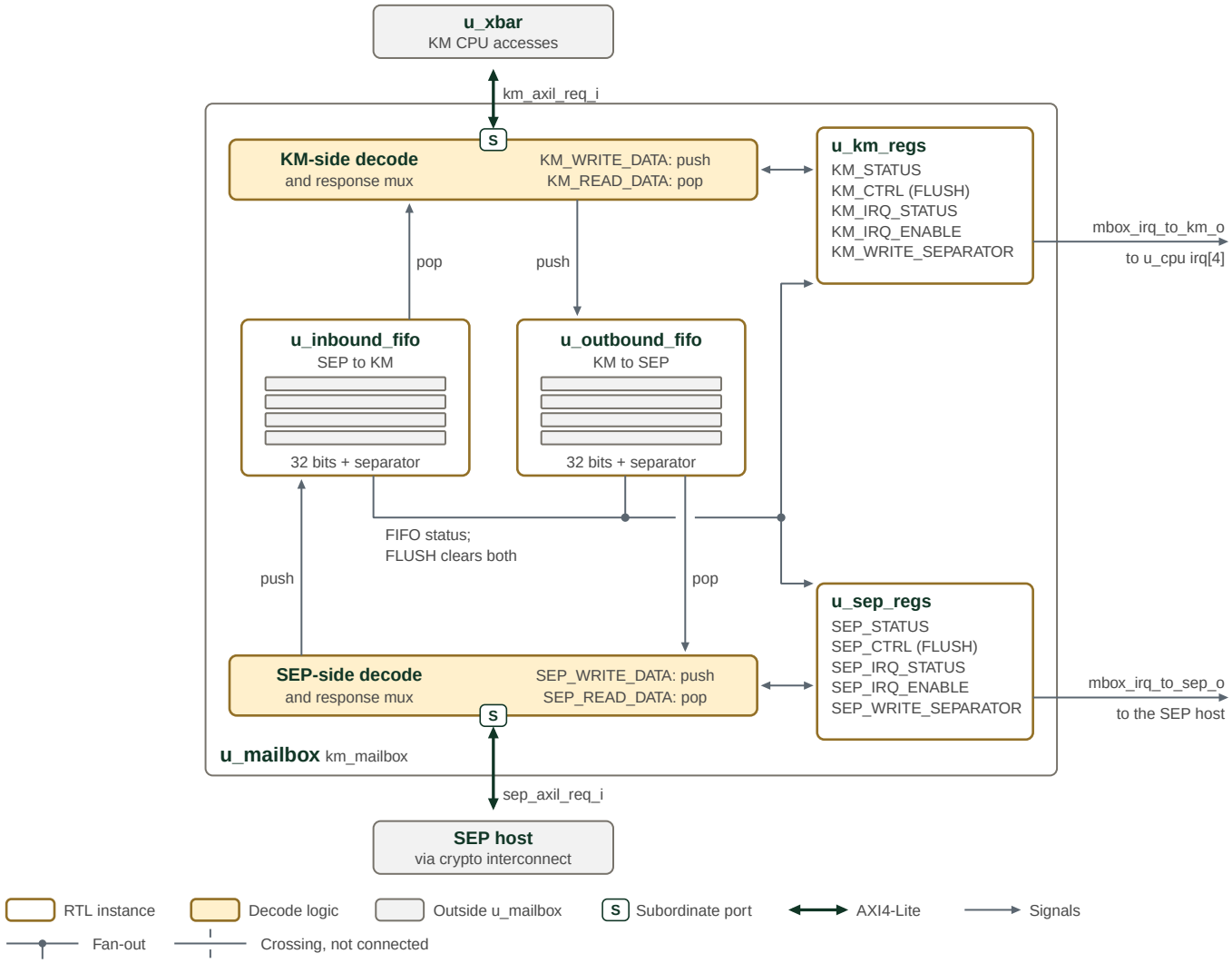


*Figure 45. Key Manager mailbox transport*

The KM CPU enters at the top and the SEP host at the bottom. Each side's decode sends its `WRITE_DATA` address to a FIFO push and its `READ_DATA` address to a FIFO pop, and every other address to its own register block. Words therefore travel upwards through `u_inbound_fifo` and downwards through `u_outbound_fifo`. Each register block reads the status of both FIFOs, and a `FLUSH` from either block clears both of them. `MAILBOX_DEPTH` sets the depth of each FIFO; SEP instantiates the KM with 16 words.

### Key Delivery

Keys reside in the KPV and reach accelerator sideload registers through the KM-mastered key path. Host-supplied material enters through mailbox commands; generated secret material is not returned through a direct host read path. The generated Memory Map defines the KM's allocated windows.

Figure 46 separates the key-delivery path on the top row from the host-accessible data path on the bottom row. This separation does not establish resistance to physical fault injection or side-channel attacks. Firmware's key-data CRC detects corruption; it is not a substitute for access control or a physical-attack guarantee.

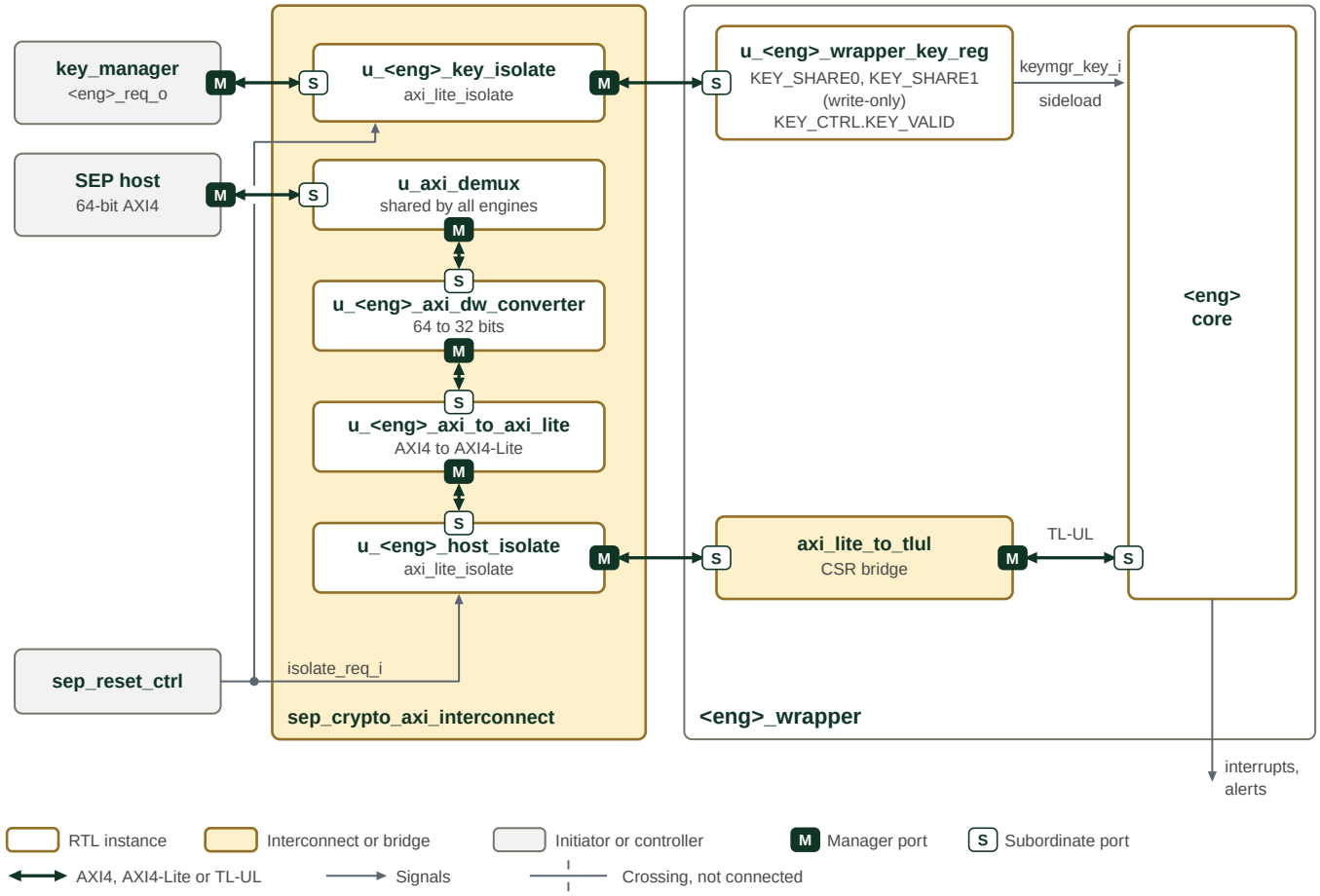


*Figure 46. Key Manager accelerator key path*

The figure shows one engine wrapper; `<eng>` stands for `otbn`, `aes`, `kmac` or `hmac`. The KM port for that engine (`otbn_req_o`, `aes_req_o`, `kmac_req_o` or `hmac_req_o`) is a point-to-point 32-bit AXI4-Lite path. It passes only through `u_<eng>_key_isolate` in `sep_crypto_axi_interconnect`, which drains the path when `sep_reset_ctrl` requests isolation, and ends on the wrapper's generated `<eng>_wrapper_key_reg` block. The `KEY_SHARE0` and `KEY_SHARE1` words are write-only, and reads return zero. The two shares and `KEY_CTRL.KEY_VALID` drive the engine's `keymgr_key_i` sideload port; the XOR of the two shares is the key. The shares are 8 words each, or 12 words for OTBN.

The SEP host reaches the engine's own registers on a separate path: the crypto interconnect demultiplexes the 64-bit AXI4 host bus in `u_axi_demux`, converts it to 32 bits in `u_<eng>_axi_dw_converter` and to AXI4-Lite in `u_<eng>_axi_to_axi_lite`, and passes it through `u_<eng>_host_isolate`, and the wrapper bridges it to the engine's TL-UL port. `isolate_req_i` from `sep_reset_ctrl` drives both isolation stages. The host path does not reach the key registers, and the KM path does not reach the engine's control registers. Adams Bridge follows the same pattern through `u_abr_key_isolate` and `abr_wrapper_key_reg`, where `sep_abr_kv_shim` turns the registers into Adams Bridge key-vault reads (see Adams Bridge Key Sideload Interfaces).

### Adams Bridge Key Sideload Interfaces

The KM provides dedicated sideload interfaces to the Adams Bridge (ABR) post-quantum engine, exposed as a single AXI4-Lite register block mapped at `0x0001_C000`–`0x0001_C7FF`. The block has two directions: four seed inputs the KM drives into ABR, and one shared-key output ABR drives back into the KM.

**Seed sideload inputs (KM → ABR).** The four seed inputs carry key material the KM side-loads into ABR (Table 377).

*Table 377. Adams Bridge seed sideload inputs*

| Interface | Width | Description |
|---|---|---|
| ML-DSA seed | 256-bit | Seed for ML-DSA key generation. |
| ML-KEM seed D | 256-bit | ML-KEM seed D. |
| ML-KEM seed Z | 256-bit | ML-KEM seed Z. |
| ML-KEM message | 256-bit | ML-KEM message input. |

Each seed uses the two-share masked layout (two key-share register banks whose XOR is the real value, plus a `key_valid` bit) and is write-only to KM firmware. The seeds are valid `dest_valid` destinations (bits 4–7), so they are targets of `CMD_KEY_TRANSFER` and `CMD_ENGINE_SHRED`, and ROM shreds them during the mutable-firmware handover. Word order matches the other engines: share word `i` is word `i` of the seed the engine consumes, as stated in ABR seed word order.

**ML-KEM shared-key output (ABR → KM).** ABR computes an ML-KEM shared key and presents it to the KM. The interface comprises a 256-bit (8-word) key register array, a `KEY_CTRL` register holding `key_valid`, and an `IRQ_STATUS` / `IRQ_ENABLE` interrupt pair.

- The key words and `KEY_CTRL.key_valid` are read-only to KM firmware: hardware sets `key_valid` and loads the key words when ABR delivers a new shared key. Firmware consumes the key by writing `KEY_CTRL = 0`, which clears `key_valid` and hardware-zeroizes (`hwclr`) the key words. Software can clear `key_valid` but cannot set it.
- `IRQ_STATUS.key_valid` is a sticky, write-one-to-clear bit (`hwset stickybit` + `onwrite=woclr`). Hardware sets it on the key-ready event; firmware clears it by writing 1. The masked interrupt to the KM CPU is `IRQ_STATUS & IRQ_ENABLE`, on a dedicated PicoRV32 interrupt (bit 5).

The KM firmware ISR acknowledges the IRQ by clearing `IRQ_STATUS` and flags the main loop; it does not consume the key. The shared key is later captured into a KPV slot in response to a SEP `CMD_ABR_SK_TRANSFER` command (see the firmware documentation).

### Key Manager CPU Interrupt Processing

The PicoRV32 receives three separate external interrupt groups:

*Table 378. KM CPU interrupt routing*

| CPU IRQ bit | Source | Service boundary |
|---|---|---|
| 3 | KMCSR | Sticky hardware events, masked by `IRQ_ENABLE`; see Table 381. |
| 4 | Mailbox | Enabled level and sticky FIFO conditions; independent of KMCSR `IRQ_STATUS`. |
| 5 | Adams Bridge shared key | Shared-key-ready status; see Adams Bridge Key Sideload Interfaces. |

KMCSR `IRQ_SET` injects its corresponding status events for ISR testing. Mailbox and Adams Bridge interrupts retain their own status/enable controls. The CPU's interrupt mask is an additional level of control. A reported event, an enabled interrupt and a CPU trap are distinct; see Unrecoverable Faults.

### Host Coordination

The SEP host submits key-load and key-transfer commands through the mailbox. The Key Manager validates the request and policy before accessing the vault or delivering key material to a cryptographic engine.

Figure 47 shows the command, validation, vault-access, and key-delivery sequence.

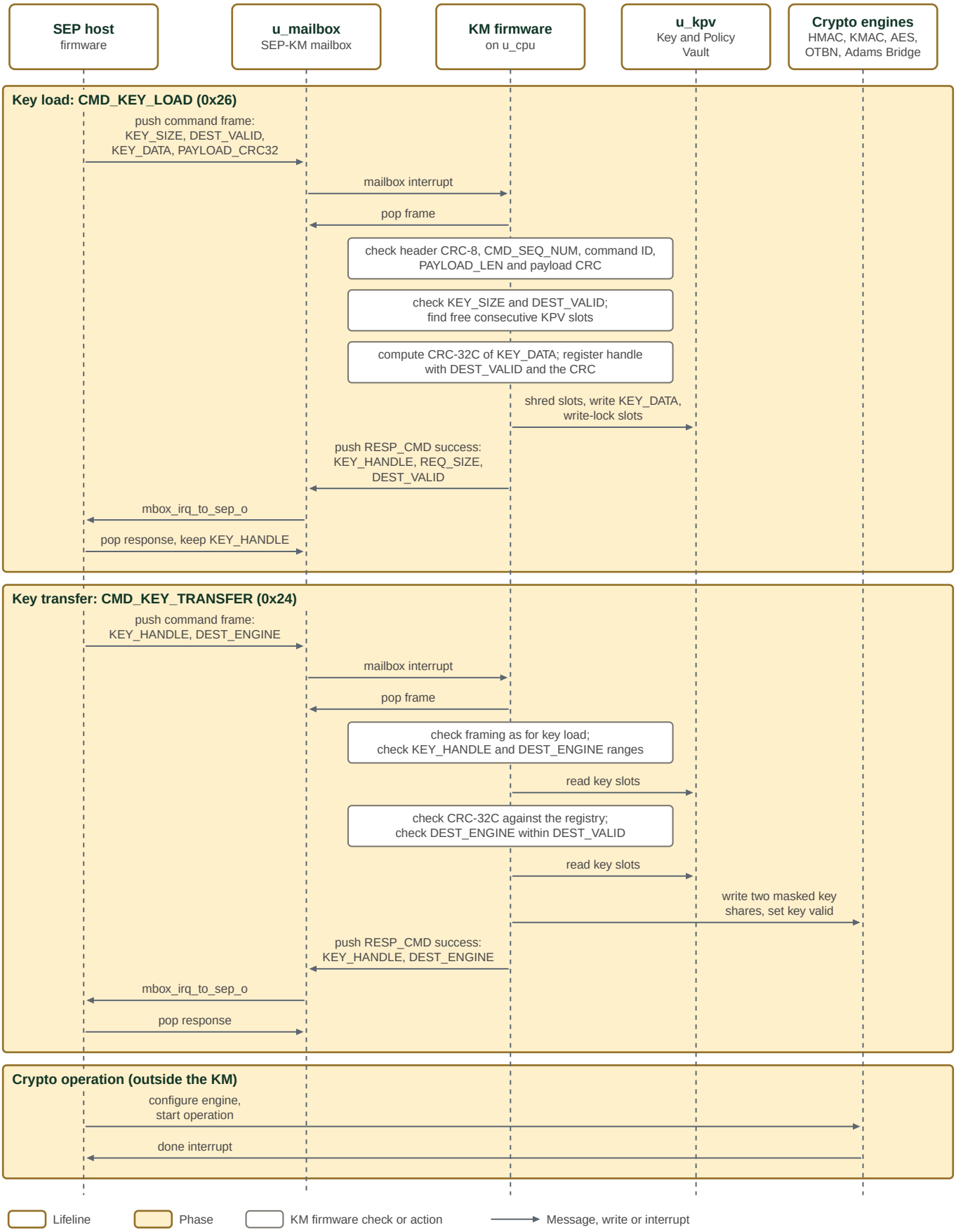


*Figure 47. SEP host and Key Manager key-load and key-transfer sequence*

Time runs downwards. Each arrow is a mailbox word push or pop, a register write or read, or an interrupt, and each white box is a check or action the KM firmware performs without leaving the CPU. The figure shows the success path. A failed check ends the command with a `RESP_CMD` error code and leaves the KPV and the key registry unchanged.

On a key load, the firmware registers the handle before it writes the KPV, and destroys the handle again if a KPV write fails. On a key transfer, the firmware reads the key slots twice: once to check their CRC-32C against the key registry, and again to write the key. Each requested engine receives the key as two key shares whose XOR is the key, the first a DRBG mask, before its key-valid bit is set. The engines are written in the order of their `DEST_ENGINE` bits, and the first failed write ends the transfer.

#### Key and Policy Vault (KPV)

The KPV exposes 64 slots through one KM-only AXI4-Lite interface. Each slot contains 512 bits (16 words) of key data and a 32-bit `CTRL` register. Multi-slot keys, such as RSA3K material, require consecutive slots; the firmware registry tracks the span and the used words in the final slot. Hardware has no span field.

*Table 379. KPV per-slot controls*

| Field | Behavior |
|---|---|
| `lock_write[0]` | Write-one-to-set. Blocks key-data writes with SLVERR; does not block CTRL writes or erase. Hardware holds it set while `seal` is set. |
| `lock_use[1]` | Write-one-to-set. Key-data reads return zero with SLVERR. Also set when an erase completes on a sealed slot. |
| `erase[2]` | Write one to overwrite all 16 words with pseudorandom data. No lock blocks erase. Poll until hardware clears the bit. If the slot is unsealed at completion, all CTRL bits clear and the slot is reusable. If sealed at completion, it is retired: `lock_write`, `lock_use` and `seal` remain set. |
| `seal[3]` | Write-one-to-set. Implies a write lock. Erase destroys the contents without making the slot reusable; taking a seal during an erase also causes retirement. |
| `[31:4]` | Reserved. |

The lock/seal bits clear on warm or cold reset. Erase uses the shared scrambler and key-data write port, so key-data accesses during an erase are not defined. The KPV driver in the Key Manager Application ROM appendix of the Programmer's Guide gives the erase sequence.

**Sealed and retired slots.** A slot with *seal* set is **sealed**: readable, but not writable, because hardware holds *lock_write* set alongside the seal. That surviving *lock_write* is evidence that the slot holds what it held when the seal was taken, which is what lets firmware that inherits the vault after a handover trust material provisioned before it. A write lock on its own would prove nothing, since an erase clears it and frees the slot to be rewritten with chosen material and re-locked.

Erasing a sealed slot is still permitted, and leaves it **retired**: *lock_write* held, *lock_use* set, *seal* still set, and the data destroyed. A retired slot can be neither read, rewritten nor reused until the next warm reset, and repeating the erase changes nothing. Revocation and reuse are therefore separate: firmware can destroy a sealed key it no longer trusts without handing the slot to whoever asked for the destruction.

An emergency wipe also releases sealed and retired slots by clearing key data and all CTRL bits. The safety of that release depends on CPU containment: a trap already halts the CPU, whereas an externally requested wipe requires firmware or host coordination (Wipe State). Do not assume that asserting `wipe_state_i` directly halts CPU execution.

CTRL resets to zero. Key-entry data flops have no reset value and power up undefined; cold or warm reset alone does not overwrite their contents.

### KPV Scrambling

The KPV supports address and data scrambling during normal operation for all key entry data reads and writes, using the same PRESENT-based approach as SRAM scrambling. When enabled, the stored representation of key data is scrambled; the KM port sees the logical (unscrambled) data on the bus, with scrambling/descrambling performed transparently by the KPV.

The KPV has a scrambler key register (32-bit) and a scrambler control register with an enable bit and a lock bit. These registers are accessible only via the KM port. The lock bit is write-one-only (0→1): software sets lock by writing 1; writing 0 has no effect; lock cannot be cleared until the next **cold** KM reset. The KPV scrambler enable and lock are state in the cold reset domain — they are preserved across a warm reset and cleared only by a cold reset. Scrambling is disabled by default after a cold reset.

When scrambling is enabled, the KPV uses the same scrambler algorithm as SRAM but sized for 1024×32 words (64 slots × 16 words). The scrambler address input is the linear register index (0..1023): *slot_index* × 16 *word_index*. Address scrambling maps the logical index to a scrambled physical storage index. Data is scrambled before storage and descrambled after retrieval. Only key entry data words are scrambled; CTRL registers are not scrambled.

The KPV scrambler key register has no reset value (powers up random/undefined for security). The scrambler control register (enable and lock) resets to 0. On an emergency wipe (rising edge of *wipe_state*), the entire KPV is reset to zero including key data, control registers, and the KPV scrambler key and control registers.

### Wipe State

Hardware forms a wipe event from `wipe_state_i | cpu_trap` and detects its rising edge. The resulting pulse clears all 1024 KPV data words, all 64 slot CTRL registers, and KPV scrambler key/enable/lock state on the clock edge. It also records the KMCSR wipe event. This hardware action covers the KPV; it does not itself scrub SRAM, mailbox data or accelerator sideload registers.

In SEP, `sep_cpu_ctrl.KM_WIPE_CTRL.wipe_state` drives the external input. Hold it high for at least one complete KM clock period. Return it to zero before requesting another rising edge; the software-controlled registered level provides the required hold. SEP does not connect a hardware tamper or alarm source to this input. A CPU trap independently triggers the same KPV wipe and asserts `unrecoverable_err_o` while the CPU is halted.

An external wipe does **not** force a CPU trap in hardware. The ROM enables the wipe interrupt and treats it as an unrecoverable fault, but that response depends on interrupt delivery and execution of its handler. Software that masks the event or replaces ROM must not assume the CPU has stopped. The host/integrator must prevent continued use of the released vault after an external wipe until recovery resets the CPU. Warm reset clears the trap and re-enters ROM; it is not itself the KPV data erase mechanism.

### Scan and Scan-Dump Protection

The complete KM is **Class 2** under SEP's scan-chain protection policy: processor, vault, KMCSR, mailbox, entropy sampler, scramblers, internal fabric, reset logic and register blocks. Including non-secret control state avoids relying on a per-register classification.

- **Test scanning is permitted only in secure test mode.** The dedicated-chain DFT controller admits test commands only with `secure_tm = 1`. At the eFuse shadow boundary, this mode replaces the four Class 1 sources (`*_UID` and `CLASS_KEY`) with valid zero values before KM captures them. Those source assets remain outside every scan chain. Public identities, lifecycle and demotion policy remain available.
- **Scan dump is never permitted.** No lifecycle state, owner or combination of `*_DBG` feature enables may turn a test command into a dump. Feature controls determine whether the chain is reachable; the command-class restriction enforces the no-dump policy. General-logic debug enablement does not grant access to the KM secret-bearing chain.

KM-local copies of the UIDs, class key and public identities clear on cold and warm reset (including firmware soft reset) and reload from the shadow bus after release. Lifecycle/demotion captures persist across warm reset. Firmware access locks are independent of scan protection.

Writable KM SRAM is a Class 2 companion memory because it can retain key, message and firmware working state. Its memory-test policy must also forbid dump. Immutable ROM cannot retain runtime secrets and is excluded. These requirements define the integration boundary; the local KM RTL alone does not establish implementation of the external DFT controller.

#### Key Manager Control and Status Registers (KMCSR)

KMCSR manages interrupts, reset requests, execution permissions, memory protection and OTP shadow readout. It is separate from the KPV and DRBG sampler banks. `km_csr` supplies the behavior around generated `km_csr_reg`; Control and Status Registers supplies every offset and field definition. The operating contracts below supplement that generated map.

The `VERSION` register (offset `0x000`) is a read-only register that reports the Key Manager hardware revision using semantic versioning. It uses the following canonical 32-bit layout, mandated so all software and testbenches decode the version identically (Table 380).

*Table 380. Key Manager VERSION fields*

| Bits | Field | Description |
|---|---|---|
| `[7:0]` | `PATCH` | Patch version component. |
| `[15:8]` | `MINOR` | Minor version component. |
| `[23:16]` | `MAJOR` | Major version component. |
| `[31:24]` | `RESERVED` | Reserved; reads as 0. |

The current hardware revision reads as version `1.0.0` (`MAJOR=1`, `MINOR=0`, `PATCH=0`) after reset with no software initialization. The register is read-only: a write completes with an AXI SLVERR response and does not change the value.

The interrupt aggregation logic collects twelve hardware event sources into a single CPU interrupt. Each source has a sticky status bit in `IRQ_STATUS` (write-one-to-clear), a mask bit in `IRQ_ENABLE`, and a software-trigger bit in `IRQ_SET` for ISR testing. The aggregated interrupt output is the logical OR of the masked status bits (`IRQ_STATUS & IRQ_ENABLE`).

Table 381 maps hardware events to the sticky interrupt-status bits.

*Table 381. Key Manager interrupt sources*

| Bit | Source | Description |
|---|---|---|
| 0 | ROM Parity Error | Parity mismatch detected on a ROM read. |
| 1 | SRAM Parity Error | Parity mismatch detected on an SRAM read. |
| 2 | ROM Write Error | A write was attempted to the read-only ROM region. |
| 3 | SRAM Write-Lock Violation | A write targeted a locked 1 KB SRAM region. |
| 4 | AXI SLVERR | An AXI4-Lite slave returned SLVERR. |
| 5 | AXI DECERR | An AXI4-Lite slave returned DECERR. |
| 6 | DRBG Error | The DRBG sampler reported a timeout or stream error. |
| 7 | Wipe State | Rising edge detected on the `wipe_state_i` input or a CPU trap. |
| 8 | OTP Change | A monitored OTP readout field's decoded value changed (see `OTP_CHANGE_STATUS`). |
| 9 | OTP Integrity Violation | OTP dual-rail integrity violation (value and complement not complementary). |

| Bit | Source | Description |
|---|---|---|
| 10 | Execute Violation | Instruction fetch from a non-whitelisted (non-executable) memory region, excluding the ROM. |
| 11 | ROM Access Violation | ROM instruction fetch or data read attempted after the ROM lockout engaged. |

Bits `[31:12]` of `IRQ_STATUS` are reserved. Two interrupt sources are not part of the KMCSR `IRQ_STATUS` aggregation and are serviced separately: the mailbox sources (FIFO data-available/space-available and overflow/underflow/flush conditions), and the Adams Bridge ML-KEM shared-key interrupt on its own PicoRV32 interrupt (bit 5, see Adams Bridge Key Sideload Interfaces).

The CPU IRQ entry vector is programmable through two registers. `IRQ_ENTRY_ADDR` holds the effective IRQ entry address and resets to the built-in default `0x0000_0010`; firmware may write any 32-bit value (the hardware performs no alignment or validity checking) and the register is always readable, returning the current effective entry address. `IRQ_ENTRY_LOCK` is a write-one-only lock bit that can only transition from 0 to 1: while it is 0 firmware may reprogram `IRQ_ENTRY_ADDR` freely; once it is 1, writes to `IRQ_ENTRY_ADDR` are ignored (the value remains readable and unchanged) and further writes to `IRQ_ENTRY_LOCK` have no effect. Both registers are in the warm reset domain: every reset that re-enters ROM (cold, external warm, and KM soft reset) restores `IRQ_ENTRY_ADDR` to `0x0000_0010` and clears `IRQ_ENTRY_LOCK`.

`SRAM_LOCK` controls writes to 32 SRAM regions of 1 KB each; see SRAM Write Locking for reset and handover policy. SRAM scrambling key/enable/lock state is cold-domain state and survives warm reset. The soft-reset code and cold-boot flag are described under Reset Conditioning.

**Execute-Permission Whitelist (NX)**

The PicoRV32 enforces a whitelist of executable memory regions on every committed instruction fetch (`mem_valid && mem_instr`). Any fetch from outside the whitelist immediately pulses `KMCSR.IRQ_STATUS.EXEC_VIOLATION`, which (if unmasked) routes to the ISR and triggers the unrecoverable fault `ROM_KM_UFAULT_EXEC`. A ROM fetch refused by the ROM lockout described below is reported on `ROM_ACCESS_VIOLATION` instead, so the two bits are disjoint.

The whitelist mode is controlled by the `KMCSR.SRAM_EXEC_MODE.enable` bit (write-1-only, warm-reset domain) (Table 382).

*Table 382. Key Manager executable regions*

| `enable` | Executable regions | Forbidden regions |
|---|---|---|
| `0` (ROM mode) | ROM (`0x0000`–`0x3FFF`) and VROM* | All SRAM; Peripheral/AXI |
| `1` (SRAM mode) | Write-locked 1 KB SRAM regions (`SRAM_LOCK.lock_bits`), VROM*, and the ROM until the lockout engages | Unlocked SRAM; Peripheral/AXI; the ROM once the lockout engages |

The fetch-region index is a plain slice of the byte address, which works only because the SRAM base (`0x0000_8000`) is aligned to the SRAM size; it is the same slice `km_sram_interface` uses to check data writes, so the two views of a region can never disagree.

- VROM is the virtual ROM window, `0x1000_0000`–`0x1000_FFFF`. Only a simulation build that defines `OCAH_KM_VROM` decodes it; there it is executable in both modes and exempt from the ROM lockout, and it carries no security requirement. Without the define the window is an unmapped address: a load or store answers DECERR, and a fetch pulses `EXEC_VIOLATION`.

For the define and the builds it is rejected in, see Key manager defines in the Integrator Guide.

`SRAM_EXEC_MODE.enable` is set by the stack-less handover assembly (`rom_handover_jump.S`) immediately before the `jalr` to mutable firmware, after SRAM write-lock regions have already been committed. This minimises the window between enabling SRAM execution and the first fetch from SRAM. The bit is write-1-only and cleared by warm reset, so the ROM-only whitelist is re-applied on every boot cycle.

#### ROM Lockout

Setting `SRAM_EXEC_MODE.enable` exchanges the ROM for write-locked SRAM rather than adding to the whitelist. Were it purely additive, mutable firmware would inherit read and fetch access to the ROM that handed control to it, leaving the ROM's key-management routines and its `rom_persist` structure reachable by code the ROM does not trust.

The exchange is staged so that the handover can complete. Writing the bit only **arms** the lockout; the lockout **engages** on the first committed instruction fetch from a write-locked SRAM address, which is the point at which mutable firmware actually begins executing. Everything the handover assembly runs after its arming store — the store itself, the register clears, the `jalr` — is ROM code, and remains legal precisely because the lockout has not engaged yet. Engaging on a fetch rather than on any ROM access also keeps the rule independent of incidental data traffic: ROM code reads `rom_persist`, which itself lives in a write-locked SRAM region.

Once engaged, the lockout blocks ROM instruction fetches and ROM data reads until the next warm or cold reset. The latch is in the warm reset domain, the same domain as `SRAM_EXEC_MODE.enable`, so the transition is one-way within a boot and every reset that re-enters ROM restores ROM access. ROM **writes** are unaffected and continue to raise `ROM_WRITE_ERR`.

**Blocked access behaviour.** A blocked access returns zeros; ROM content is never delivered. What follows depends on the access type:

- A blocked **data read** raises `ROM_ACCESS_VIOLATION` and nothing else. Zero is a legitimate data value, so no decode and no trap occur.
- A blocked **instruction fetch** raises `ROM_ACCESS_VIOLATION` and additionally feeds an all-zeros instruction stream, which the PicoRV32 `CATCH_ILLINSN` logic traps as an illegal instruction.

The two reports are ordered by causality rather than by arbitration. The violation is combinational in the cycle of the blocked access, whereas the illegal instruction cannot be detected until the zero word has been returned and decoded, which is strictly later. Since `IRQ_STATUS` is sticky, the violation bit is always recorded first.

Which mechanism **acts** depends on the interrupt mask, and after handover only one does. Handover leaves all CPU interrupts masked, so the illegal instruction takes the PicoRV32 trap state rather than an interrupt: the core halts, `unrecoverable_err_o` asserts, and the KPV is wiped. The `ROM_ACCESS_VIOLATION` interrupt is inert in that state — it sets its sticky bit but has nowhere to vector, because the ROM ISR it would have vectored to is itself in the ROM the lockout has just revoked. A real post-handover breach therefore halts the KM without emitting a mailbox fault code; `ROM_KM_UFAULT_ROM_ACCESS` is reportable only while the ROM is still live (via `IRQ_SET`) or through a vector that mutable firmware installed itself.

Both mechanisms are retained because they do different jobs. The violation bit is the diagnostic that distinguishes a lockout breach from any other illegal-instruction halt; the trap is the containment, and it requires no firmware cooperation, no enabled interrupt, and no reachable handler.

#### Error Events

#### Unrecoverable Faults

These events cause the CPU to halt when the corresponding interrupt is unmasked and the ISR determines recovery is not possible. They include, but are not limited to, the following conditions:

- CPU trap events
- ROM or SRAM parity errors
- Invalid writes to ROM or SRAM

- AXI bus faults (SLVERR or DECERR)
- Instruction fetch from a non-whitelisted (non-executable) memory region
- ROM instruction fetch or data read after the ROM lockout engaged
- Any other unhandled exception or fault

The PicoRV32 includes a trap state that asserts a trap signal. When the CPU enters this state execution halts. The CPU will enter this state under the following conditions:

- An EBREAK or other illegal instruction is encountered *and* the associated interrupt is masked.
- The CPU encounters a misaligned access *and* the associated interrupt is masked.
- An EBREAK or other illegal instruction is encountered within an ISR.

If an ISR determines that an error event is not recoverable, it will call the EBREAK instruction to trap and halt the core. The CPU trap signal is passed out of KM to indicate an unrecoverable fault. This signal indicates the KM has encountered an unrecoverable fault and is no longer executing (it will not respond to further commands). The signal remains asserted for as long as the KM remains in the trap state. The trap is cleared by resetting the KM CPU — either a cold reset or a warm reset (the PicoRV32 core is in the warm reset domain) restarts the core and clears this signal.

**Recoverable Faults**

These events do not cause the CPU to halt. They include:

- Receiving corrupt or malformed command messages.
- Detection of possible tampering or attack conditions that are not immediately terminal.
- Any one of the previously listed error interrupts from which the corresponding ISR is able to gracefully recover.

The KM provides an output signal that indicates when a recoverable fault has been successfully handled. This signal is driven by a dedicated register bit in the KMCSR. When an ISR recovers gracefully from an error, it sets this register bit. The bit remains set until the SEP sends a firmware command to the KM to clear it (KM firmware clears the register bit in response to that command).

**DRBG Sampler**

The sampler bridges a CPU `DATA` read to the selected 32-bit entropy stream. It is an AXI4-Stream sink: `TREADY` is an output, and `TVALID`, `TDATA[31:0]` and `TSTRB[3:0]` are inputs. Availability is not guaranteed; a pending read waits for a word or an enabled timeout.

**Reads, prefetch and register access**

A `DATA` read consumes a prefetched word if one is available. Otherwise the sampler asserts `TREADY` and assembles four valid bytes before completing the AXI read. `TSTRB` selects bytes to pack in beat order, walking lanes 0 through 3; partial beats require further handshakes. Bytes beyond the four needed for the current word are not carried into the next word.

`CFG.PREFETCH` enables a single prefetched word. Consuming it through `DATA` invalidates it; prefetch alone does not increment the direct-read success counter. Disabling prefetch clears its data/valid state. The read-only `PREFETCH_DATA` register exposes the stored word for debugging and does not consume it, so repeated debug reads are not fresh random samples.

*Table 383. DRBG sampler register behavior*

| Register or field | Access and meaning |
|---|---|
| `DATA` | Read obtains a sample; writes return SLVERR. |
| `CFG.PREFETCH[0]` | Read/write; reset 0 (disabled). |
| `CFG.TIMEOUT[31:16]` | Read/write active-read wait budget in clocks; reset 256. The counter loads `TIMEOUT - 1` and times out after exactly `TIMEOUT` active cycles. Zero disables timeout. Background prefetch has no timeout. |
| `STATUS.DRBG_READY[0]` | Read-only indication of input `TVALID`. |
| `STATUS.PREFETCHED[1]` | Read-only indication that a prefetched word is available. |
| `STATUS.TIMEOUT_ERR[2]` | Sticky timeout indication; write one to clear. |
| `STATUS.STREAM_ERR[3]` | Sticky stream-protocol error; write one to clear. |
| `STATUS.COUNT_BAD[15:8]` | Timeout/stream-error count, saturating at `0xFF`. Write nonzero to clear; zero has no effect. |
| `STATUS.COUNT_GOOD[31:16]` | Counts only reads completed through `ST_RESPOND`, excluding prefetched words, saturating at `0xFFFF`. Write nonzero to clear; zero has no effect. |
| `PREFETCH_DATA` | Read-only debug view; writes return SLVERR. |

Because prefetched reads bypass `ST_RESPOND`, `COUNT_GOOD` is not a total of all `DATA` reads when prefetch is enabled.

### Errors and reset

Timeout completes an active read with SLVERR and zero data. The protocol monitor detects a different event: previous-cycle `TVALID=1`, `TREADY=0`, followed by `TVALID=0` (withdrawal before handshake). It sets `STREAM_ERR`, increments `COUNT_BAD` and pulses the KMCSR DRBG error source even when no read is active. If the active FSM is requesting or assembling a word, it also returns SLVERR with zero data. A prefetch in progress is discarded. A `TVALID` drop after a completed handshake is not this violation.

The sampler is in the KM warm-reset domain, not the TRNG reset domain. Quiesce KM before resetting the TRNG. Resetting the producer alone can withdraw an unaccepted beat or interrupt byte assembly; it does not clear partial bytes held by the sampler. Holding KM in software reset clears sampler control/assembly state before the source restarts.

Warm or cold KM reset restores configuration defaults, counters, error bits, valid flags and request/assembly control. Data registers (prefetch data, assembly storage and the response slot) have no reset value; invalidating their control state is not a guaranteed data scrub.

*Table 384. Key Manager DRBG sampler states*

| State | Active DATA-read behavior |
|---|---|
| `ST_IDLE` | Returns an available prefetch without entering `ST_RESPOND`; otherwise enters `ST_REQUEST` for a pending read. |
| `ST_REQUEST` | Asserts `TREADY`. A full word advances to `ST_RESPOND`; partial data advances to `ST_BYTE_ASSEMBLY`. |
| `ST_BYTE_ASSEMBLY` | Keeps `TREADY` asserted and packs valid bytes until four are available. |
| `ST_RESPOND` | Completes the assembled read with OKAY and increments `COUNT_GOOD`; returns to idle. |
| `ST_TIMEOUT` | Completes with SLVERR and zero data after the active wait budget expires; returns to idle. |
| `ST_STREAM_ERR` | Completes with SLVERR and zero data after a detected withdrawal during request/assembly; returns to idle. |

A separate prefetch FSM uses the request/assembly states. It accepts beats only when there is no active CPU `DATA` read. In the active request/assembly states, a stream error takes priority over timeout, and timeout takes priority over accepting a completing beat.

#### OTP Inputs and Readout

The KM consumes OTP-derived material over a private hardwired bundle so the values cannot be spoofed by software. SEP dual-rail-encodes each 256-bit field at the SEP-to-KM boundary (`{~value, value}`) from the sensed eFuse shadow:

- the fuse-provisioned secrets — `CHIPLET_UID`, `SIP_UID`, `SYS_UID`, `CLASS_KEY`;
- the three 256-bit public identities — `SEP_CHIPLET_ID`, `SEP_SIP_ID`, `SEP_SYS_ID` — one per owner;
- the life-cycle state.

**UID delivery and protection.** The UIDs are held in a KM-local copy. The KM's own warm-domain read-lock protects that copy at mutable-firmware handover (see UID provisioning and access) so mutable firmware cannot read it. This KM-local lock is separate from, and additional to, the eFuse controller's own locks.

**Public identity delivery.** The three public identity fuses take the same dual-rail encode as the secrets. They remain on the KM port in secure test mode: the eFuse controller blanks only the four secrets on the hardware shadow output. The eFuse per-field read-lock bits (`LOCKS[59]`, `[61]`, `[63]`) hide software and JTAG shadow reads; they do not blank the hardwired KM port. The KM's own `OTP_READ_LOCK` and `OTP_READ_LOCK_COLD` bits zero the KMCSR readout words only.

**Life-cycle state.** The life-cycle state is hardwired into the KM as authoritative, software-readable status. Being non-secret it is neither locked nor wiped. The key registry does not bind keys to it.

**eFuse access path.** The KM also has a programmatic (AXI-Lite) path to the eFuse controller — for example to request a read or a program operation. That path is treated exactly like a system-host access: same access filtering, and it honors the eFuse controller's native per-field read/write locks, so the KM gains no read privilege over the fuse array beyond the SEP host's.

**Demotion and PCR.** The owner demotion state is hardwired from the Life Cycle Controller (LCC). A PCR Vault (PCRV) and PCR-change interrupt are not present in this integration.

**KMCSR readout registers.** The OTP-derived fields above are exposed to KM firmware in the KMCSR region as hardware-driven, read-only registers. Each field is delivered to the KMCSR as dual-rail complementary values (`{~value, value}`) and decoded within the KMCSR, which continuously verifies the complement relationship.

Table 385 lists the OTP-derived fields exposed to Key Manager firmware.

*Table 385. Key Manager OTP readout registers*

| Register | Width | Contents |
|---|---|---|
| `OTP_LIFE_CYCLE` | 8-bit | Differential-encoded life-cycle state `{~lc[3:0], lc[3:0]}`. |
| `OTP_DEMOTION_STATE` | 4-bit | Two 1-bit demotion flags, each differentially encoded into 2 bits: `{demote_2[1:0], demote_1[1:0]}`. |
| `OTP_CHIPLET_UID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit CHIPLET_UID dual-rail words. |
| `OTP_SIP_UID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit SIP_UID dual-rail words. |
| `OTP_SYS_UID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit SYS_UID dual-rail words. |
| `OTP_CLASS_KEY_VAL/CPL_0..7` | 8 × 32-bit | 256-bit CLASS_KEY dual-rail words. |
| `OTP_SEP_CHIPLET_ID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit SEP_CHIPLET_ID dual-rail words. |
| `OTP_SEP_SIP_ID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit SEP_SIP_ID dual-rail words. |
| `OTP_SEP_SYS_ID_VAL/CPL_0..7` | 8 × 32-bit | 256-bit SEP_SYS_ID dual-rail words. |
| `OTP_READ_LOCK` | 9-bit | Write-one-to-set read-lock bits (warm reset domain), one per field. Cleared by warm or cold reset. |

| Register | Width | Contents |
|---|---|---|
| `OTP_READ_LOCK_COLD` | 9-bit | Write-one-to-set read-lock bits (cold reset domain), one per field. Survives warm reset; cleared only by cold reset. Effective lock = `OTP_READ_LOCK` \| `OTP_READ_LOCK_COLD`. |
| `OTP_CHANGE_STATUS` | 9-bit | Sticky write-1-to-clear flags: set when a monitored field's decoded value changes. |

**Dual-rail integrity.** The hardware continuously checks that each value word is the bitwise inverse of its complement word. Any violation raises the `OTP_SIGINT` interrupt, which ROM firmware treats as an unrecoverable fault. This applies to the public identity fields as much as to the secrets.

**Change detection.** When a monitored field's decoded value changes, the hardware sets the corresponding `OTP_CHANGE_STATUS` bit and raises the `OTP_CHANGE` interrupt. ROM firmware reads the status, invokes an overridable change hook, and clears the flag. Spurious change events are suppressed until the capture history is valid after cold or warm reset.

#### Unique Identifiers (UID) — provisioning and access

The KM receives three fuse-provisioned Unique Identifiers — `CHIPLET_UID`, `SIP_UID`, and `SYS_UID` (256 bits each) — provisioned respectively by the chiplet creator, the SiP owner, and the system owner. ROM does not derive keys from them.

The CPU services mailbox commands from ROM and supports loading and executing SRAM firmware. Before handing control to that firmware, ROM applies the OTP read locks and wipes secrets from the shared execution environment. A warm reset returns the CPU to ROM, where the handover protections are re-established.

Concretely, this safeguard is the `OTP_READ_LOCK` register: one write-one-to-set bit per OTP readout field (life-cycle, demotion, the four 256-bit secrets, and the three 256-bit public identities). Setting a bit causes the hardware to return zero for that field's value and complement words. ROM applies the lock using a triple-write convention (three consecutive writes) so it commits even under a single-bit glitch. `OTP_CHANGE_STATUS` shares the same warm reset domain for the same reasons.

ROM does not lock the public identity fields at handover, since concealing them buys nothing. Their lock bits suppress KMCSR readback like those of the secret fields.

A parallel **cold-reset-domain** register, `OTP_READ_LOCK_COLD`, provides the same write-one-to-set, per-field locking semantics but survives warm reset (including soft reset) and is cleared only by cold reset. The effective read-lock for each field is the OR of the corresponding bits in `OTP_READ_LOCK` and `OTP_READ_LOCK_COLD`; either register alone is sufficient to suppress CSR readback. The SEP host can set cold-domain lock bits at any time by issuing `CMD_OTP_READ_LOCK_COLD` (0x28) over the KM mailbox. This allows the SEP host to establish read-locks that persist across warm resets for the entire duration of a power cycle, independently of the warm-reset lock applied by ROM firmware.

#### CRC Accelerator

The KM CPU is fitted with a CRC accelerator attached through the PicoRV32 PCPI (Pico Co-Processor Interface). It provides hardware versions of the CRC functions the firmware uses on its high-value integrity paths — message header CRC-8/ROHC, message payload CRC-32C, and key-data integrity CRC-32C — so these computations consume far less CPU time than a software loop. The accelerator is always present in the KM build; firmware invokes it unconditionally and does not retain a software fallback path.

The accelerator exposes three R-type custom instructions on the PicoRV32 `custom0` opcode. Each instruction takes two operands — the current CRC chaining state (`rs1`) and the next input unit (`rs2`) — and writes the updated chaining state to `rd`.

Table 386 lists the instruction encodings and input units for the hardware CRC updates.

*Table 386. CRC Accelerator Instructions*

| Instruction | funct7 | funct3 | opcode | Input unit |
|---|---|---|---|---|
| CRC-32C word update | 7'b0101100 | 3'b000 | custom0 | rs2[31:0], four bytes little-endian |
| CRC-32C byte update | 7'b0101100 | 3'b001 | custom0 | rs2[7:0] (rs2[31:8] ignored) |
| CRC-8/ROHC byte update | 7'b0101100 | 3'b010 | custom0 | rs2[7:0] (rs2[31:8] ignored) |

Behavioral properties:

- The CRC-32C word-update instruction consumes its 32-bit operand as four bytes in little-endian order ([7:0], [15:8], [23:16], [31:24]).
- The CRC-32C instructions expose the internal running remainder **before** the final XOR. Firmware seeds the chaining state with 0xFFFF_FFFF and applies the final XOR only when it needs the externally visible CRC value, so word updates, byte updates, and intermediate states chain freely.
- For non-word-sized buffers, firmware processes whole words with the word-update instruction and any remaining 1–3 tail bytes with the byte-update instruction; no software tail path is used.
- The CRC-8/ROHC instruction carries its 8-bit state in rd[7:0] and zero-extends rd[31:8].
- Empty-input results are identical to the software definitions: with no input units consumed, the seeded state (after the firmware's final XOR for CRC-32C) is the empty-input CRC.

The three instructions share a single byte-update datapath. CRC-32C word mode runs four byte-update cycles per operation; the two byte modes run one cycle each. The datapath uses reflected CRC update behavior with the reflected polynomial 0x82F6_3B78 for CRC-32C and 0x0000_00E0 for CRC-8/ROHC (the reflection of the 0x07 CRC-8/ROHC polynomial), and never applies the CRC-32C final XOR. The CRC parameter definitions (init, reflection, and final XOR) are given in the firmware CRC function library documentation.

Any other encoding in the custom0 CRC instruction family is not recognized by the accelerator: it does not assert the PCPI ready/wait/write handshakes, so PicoRV32's normal illegal-instruction handling is preserved. The feature introduces no new top-level KM ports or CSRs, and the PicoRV32 built-in multiply/divide support remains enabled and unchanged.

#### ROM and SRAM Error Detection

ROM and SRAM use one odd-parity bit per byte: four parity bits accompany each 32-bit data word. Checking cannot be disabled and performs no correction. Interrupt enables control reporting to the CPU; masking an event does not repair the affected data or make it safe to use.

#### Parity Error Protection

Parity bits are stored out-of-band, outside the scrambled data word. This keeps parity storage separate from the scramble/descramble data path.

#### SRAM Write Locking

The SRAM interface incorporates a write-locking mechanism to designate specific 1 KB memory regions as read-only; 32 regions cover the whole 32 KB SRAM, one per SRAM_LOCK bit. This feature is primarily designed to protect dynamically loaded KM firmware from unauthorized modification.

Control of this functionality resides within the Key Manager Control and Status Registers (KMCSR). Once a memory region is locked, it cannot be unlocked within the current boot session. It will remain in the read-only state until the

next warm or cold reset of the KM. The write-lock state is in the warm reset domain and is cleared on every warm or cold reset.

Protection across warm resets is maintained by the ROM, which is responsible for re-applying the full write-lock policy on every boot (both cold and warm) before handing control to mutable firmware. Because the KM CPU always re-enters ROM after a warm reset, mutable firmware never executes against an unlocked SRAM region.

#### ROM Warm-Persistent SRAM Region

The highest 1 KB of KM SRAM (region 31, `0x0000_FC00`–`0x0000_FFFF`) are reserved as a ROM-managed warm-persistent data structure (`rom_persist`). The mutable-firmware load area, stack, and BSS/data occupy the SRAM below it.

This region is owned exclusively by ROM firmware: it holds ROM state that must survive warm reset and is not general-purpose scratch for mutable (SRAM-loaded) firmware. Mutable firmware must not place its own data here, and once ROM write-locks the region it cannot write it at all (see **Write-lock policy** below).

SRAM contents survive warm or soft reset. Reset of the memory interface does not itself specify an array erase. ROM carries state across warm resets and initializes this structure on cold boot; this is volatile SRAM retention, not non-volatile storage.

**Cold initialization.** On every cold boot, `rom_boot_init()` detects a cold boot via the `COLD_BOOT_DONE` flag (reads 0 before it is set at the end of boot) and calls `rom_persist_cold_init()`, which writes known-zero values to all fields. On a warm or soft reset, `COLD_BOOT_DONE` already reads 1 and `rom_persist_cold_init()` is skipped, preserving the contents of the region.

**Contents.** The `rom_persist_t` structure currently contains one field (Table 387).

*Table 387. Key Manager warm-persistent SRAM contents*

| Field | Type | Description |
|---|---|---|
| `sram_fw_size` | `uint32_t` | Byte count of the mutable firmware image currently loaded in SRAM (`FW_WORDS x 4`). Zero means no mutable firmware is present. Written by `CMD_SRAM_LOAD_EXEC` on a successful load and verified by `CMD_SRAM_EXEC` before jumping. |

**Write-lock policy.** Before handing control to mutable (SRAM-loaded) firmware, ROM must call `rom_persist_lock()`, which sets SRAM_LOCK bit 31 via a triple write for glitch resistance. After locking, any write into `0xFC00`–`0xFFFF` is silently dropped and a `SRAM_WRITE_LOCK_VIOLATION` IRQ is raised. The lock is cleared on the next warm or cold reset; ROM re-establishes it before transferring control to mutable code.

#### Mutable Firmware Model

The KM ROM can load a mutable image into SRAM and hand control to it, through the `CMD_SRAM_LOAD_EXEC` and `CMD_SRAM_EXEC` mailbox commands. The image is loaded at `0x8000`, which is also its entry point, and must fit below a static load ceiling aligned to a 1 KB write-lock region, so the image never shares a region with the ROM stack or data. The load bounds, the key shred, the KPV read-lock, the image write-lock, the mailbox flush, the SRAM scramble and the jump are ROM behaviour, defined in the SRAM Firmware Handover sequence of the Key Manager Application ROM appendix in the Programmer's Guide.

The handover relies on the following hardware behaviour:

- `SRAM_LOCK` is in the warm-reset domain. A warm reset clears the image write-lock, and ROM re-applies it before every handover, including a `CMD_SRAM_EXEC` restart.
- `IRQ_ENTRY_ADDR` and `IRQ_ENTRY_LOCK` are warm-reset-domain registers. Every reset that re-enters ROM (cold, external warm, and KM soft reset) restores the vector to the ROM default (`0x10`) and clears the lock, so ROM never executes

with a vector chosen by mutable firmware.

- KPV key data has no warm reset and is descrambled on KM-port reads. `lock_use` is independent of `lock_write` and makes KM-port key reads return zero.
- The crypto-engine sideload key registers lie outside the SRAM address space, so scrambling SRAM does not reach them.
- Writes to a write-locked SRAM region are dropped and raise `SRAM_WRITE_LOCK_VIOLATION`, which is sticky until cleared.

#### SRAM Scrambling

Both the address as well as data can optionally be scrambled. The goal of the scrambler is not secure data encryption but rather basic obfuscation to impede simple memory dumping by an attacker aiming to recover secrets. The algorithm would be straight forward to break by a trained cryptanalyst. The implementation is intentionally very simple so that the timing overhead on memory read and write transactions is minimal.

The scrambler is disabled by default, but a configuration register field can be set to turn this feature on after booting and the scrambling key is installed. When it is on, all write data transactions will have both the address and data scrambled before performing a write to memory. On read operations, the address will be scrambled in the same fashion as for writes so we have an equivalent address mapping. The read data however must be descrambled. Address data tweaks the key, which means it is mixed in to ensure that the same data at different address locations has a different scrambled value.

The KM ROM writes DRBG output into the key register and then enables and locks the scrambler; the Memory Scrambler Provisioning section of the Key Manager Application ROM appendix in the Programmer's Guide gives the order.

The scrambler IP consists of a generic top-level module (`scrambler`) and a set of pre-configured size-specific wrappers that fix the address width to match a particular SRAM depth. The generic module selects the correct address scrambling function at elaboration time through generate blocks keyed on `ADDR_WIDTH`. The `scrambler_addr_tweak` submodule expands and mixes the memory address into a 32-bit round key. All cryptographic primitives (S-boxes, permutations, and address scramble functions) are declared as `automatic` functions in `scrambler_pkg`.

#### Parameters

The scrambler module accepts three parameters. Only 32-bit data widths are supported in the current implementation.

The parameters and ports of `scrambler` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Port Interface

The scrambler is purely combinational. It has no clock or reset ports.

#### Data Scrambling Algorithm

The scrambling algorithm is an extremely rudimentary, round reduced, block cipher inspired by the PRESENT cipher. PRESENT works on 64 bit blocks, while this scrambler works on 32-bit blocks. PRESENT operates on 32 repeated rounds, while this scrambler performs only a single round. The player (permutation function) is also altered to accommodate the reduced block size i.e. word length.

The data path implements the PRESENT 4-bit S-box (`sbox4`) from `scrambler_pkg`. Eight parallel S-box instances process the 32-bit word one nibble at a time. The S-box mapping is shown in Table 388.

*Table 388. SRAM scrambler S-box*

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S(x) | C | 5 | 6 | B | 9 | 0 | A | D | 3 | E | F | 8 | 4 | 7 | 1 | 2 |

The inverse S-box (`ibox4`) reverses the mapping and is used on the descramble path.

The permutation layer follows the PRESENT structure with progressive offsets of 8 bits rather than 16. Index *i* indicates the input bit position and *P(i)* indicates the output bit position.

Table 389 maps input bits 31 through 16 to their output positions.

*Table 389. SRAM scrambler permutation for bits 31 to 16*

| i | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 0 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 1 | 5 | 9 | 13 | 17 | 21 | 25 | 29 |

Table 390 maps input bits 15 through 0 to their output positions.

*Table 390. SRAM scrambler permutation for bits 15 to 0*

| i | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 2 | 6 | 10 | 14 | 18 | 22 | 26 | 30 | 3 | 7 | 11 | 15 | 19 | 23 | 27 | 31 |

**Scrambling Procedure**

First the tweaked key is xor'ed with the input write data. The output is split into 8 4-bit nibble lanes and each is fed into an sbox function. The resulting values are concatenated and fed into the player permutation function to form the final scrambled data value which is written to memory.

**Descrambling Procedure**

The descrambling process is exactly the reverse as scrambling. The scrambled data value is read from memory and we first perform the iplayer inverse permutation operation. This is followed by inverse sbox functions applied to each 4-bit lane, with the outputs concatenated. This word is then xor'ed with the tweaked key to recover the unscrambled read data.

**Address Tweaking:**

```
Rkey = scrambler_key[31:0] ^ {addr[N-1:0] repeated to fill 32-bits}
```

**Scrambling:**

```
Write_data_sbi = Write_data ^ Rkey
Write_data_sb = {
    sbox(write_data_ak[31:28]),
    sbox(write_data_ak[27:24]),
    sbox(write_data_ak[23:20]),
    sbox(write_data_ak[19:16]),
    sbox(write_data_ak[15:12]),
    sbox(write_data_ak[11:8]),
```

```
    sbox(write_data_ak[7:4]),
    sbox(write_data_ak[3:0])}
Scrambled_write_data = player(Write_data_sb)
```

**Descrambling:**

```
Read_data_sbi = iplayer(Scrambled_read_data)
Read_data_sbo = {
    ibox(Read_data_sbi[31:28]),
    ibox(Read_data_sbi[27:24]),
    ibox(Read_data_sbi[23:20]),
    ibox(Read_data_sbi[19:16]),
    ibox(Read_data_sbi[15:12]),
    ibox(Read_data_sbi[11:8]),
    ibox(Read_data_sbi[7:4]),
    ibox(Read_data_sbi[3:0])}
Read_data = Read_data_sbo ^ Rkey
```

#### Address Tweaking

As previously mentioned, address tweaking is necessary so that the same data at different addresses scrambles to unique values. Tweaking is performed by simply xor mixing the address with the random scrambler key to form the round key. If the address is less than 32 bits wide, then we just repeatedly concatenate the address with itself until we have 32 bits, truncating any excess bits.

The `scrambler_addr_tweak` submodule implements this expansion. For power-of-two-aligned widths (8 and 16), the address is replicated exactly to fill 32 bits. For non-power-of-two widths, the address is concatenated three times and the upper portion of the third copy is truncated to produce exactly 32 bits. Table 391 lists the expansion formula used for each supported width.

*Table 391. SRAM scrambler address expansion*

| ADDR_WIDTH | Expansion to 32 bits |
|---|---|
| 8 | `{addr, addr, addr, addr}` |
| 9 | `{addr, addr, addr, addr[8:4]}` (9+9+9+5 = 32) |
| 10 | `{addr, addr, addr, addr[9:8]}` (10+10+10+2 = 32) |
| 11 | `{addr, addr, addr[10:1]}` (11+11+10 = 32) |
| 12 | `{addr, addr, addr[11:4]}` (12+12+8 = 32) |
| 13 | `{addr, addr, addr[12:7]}` (13+13+6 = 32) |
| 16 | `{addr, addr}` |
| 32 | Identity (no expansion). |

The expanded address is XORed with `scrambler_key_i` to produce the 32-bit `round_key_o`.

#### Address Scrambling

Address scrambling is done in a similar fashion as data scrambling with an arbitrary permutation function. The address

bits are first xor'ed with the key bits according to the address width. The result is then split into 3 or 4 bit lanes, and passed through one sbox per lane. Finally the sbox results are concatenated and fed through a linear permutation network. There is *no descrambling operation on the address*, only scrambling.

Each supported address width has a dedicated `addr_scrambleN` function in `scrambler_pkg`. These functions partition the address-key-XOR result into lanes of 3 or 4 bits, apply `sbox3` (Pyjamask) or `sbox4` (PRESENT) to each lane, concatenate the S-box outputs, and pass the result through the corresponding `permN` fixed permutation. The lane partitioning varies by width to ensure the full address is covered without padding.

Table 392 identifies the S-box lanes and permutation used at each address width.

*Table 392. SRAM scrambler address lanes*

| Width | S-box Lane Partition | S-box Types | Permutation |
|---|---|---|---|
| 6 | `[5:3], [2:0]` | `sbox3`, `sbox3` | `perm6` |
| 7 | `[6:3], [2:0]` | `sbox4`, `sbox3` | `perm7` |
| 8 | `[7:4], [3:0]` | `sbox4`, `sbox4` | `perm8` |
| 9 | `[8:6], [5:3], [2:0]` | `sbox3` x3 | `perm9` |
| 10 | `[9:6], [5:3], [2:0]` | `sbox4`, `sbox3` x2 | `perm10` |
| 11 | `[10:8], [7:4], [3:0]` | `sbox3`, `sbox4` x2 | `perm11` |
| 12 | `[11:8], [7:4], [3:0]` | `sbox4` x3 | `perm12` |
| 13 | `[12:9], [8:6], [5:3], [2:0]` | `sbox4`, `sbox3` x3 | `perm13` |

### Byte-Wise Scrambling Mode

When the `BYTE_WISE` parameter is set to 1, the permutation layer operates on each byte independently rather than across the full 32-bit word. On the write path, the eight S-box outputs are grouped into four byte lanes and each lane is permuted with `perm8`. On the read path, `iperm8` is applied per byte lane before the inverse S-box stage. The S-box and round-key XOR stages are identical regardless of the `BYTE_WISE` setting.

This mode is used in the Key Manager SRAM interface where byte-level write strobes require that scrambling of one byte lane does not depend on data in other lanes.

### Instantiation Variants

Five pre-configured modules are provided, each fixing the `ADDR_WIDTH` parameter for a specific SRAM depth. All variants default to `DATA_WIDTH` = 32 and `BYTE_WISE` = 0, though the `BYTE_WISE` parameter can be overridden at instantiation.

Table 393 relates each preconfigured module to its SRAM depth and address-scramble function.

*Table 393. SRAM scrambler module variants*

| Module | ADDR_WIDTH | SRAM Depth | Address Scramble Function |
|---|---|---|---|
| `scrambler_512x32` | 9 | 512 words | `addr_scramble9` |
| `scrambler_1024x32` | 10 | 1024 words | `addr_scramble10` |
| `scrambler_2048x32` | 11 | 2048 words | `addr_scramble11` |
| `scrambler_4096x32` | 12 | 4096 words | `addr_scramble12` |
| `scrambler_8192x32` | 13 | 8192 words | `addr_scramble13` |

### Integration in the Key Manager

The Key Manager SRAM interface (`km_sram_interface`) instantiates two `scrambler_8192x32` instances with `BYTE_WISE` = 1, whose 13-bit address covers the 8192 words of the 32 KiB SRAM. The first instance (`u_req_scrambler`) scrambles the write data and address on the request path. The second instance (`u_rsp_descrambler`) descrambles read data on the response path using the address captured when the SRAM accepts the read.

The scrambler is controlled through the `SCRAMBLER_KEY` and `SCRAMBLER_CTRL` registers in the Key Manager CSR block (`km_csr`). The `SCRAMBLER_CTRL` register has two fields: `enable` at bit 0 and `lock` at bit 1. When `lock` is written to 1, the transition is permanent until the next reset. Once locked, the key register reads as zero, writes to it are ignored, and the scrambler is forced into the enabled state. This prevents firmware from disabling scrambling or reading back the key after provisioning.

A bypass multiplexer gates the scrambled data path. When `scrambler_en_i` is deasserted, addresses and data pass through unmodified. Parity is always computed on the plaintext data before scrambling, ensuring that the parity check on the read path operates on the descrambled value.

Figure 48 shows the address and data scrambling paths at the SRAM interface.

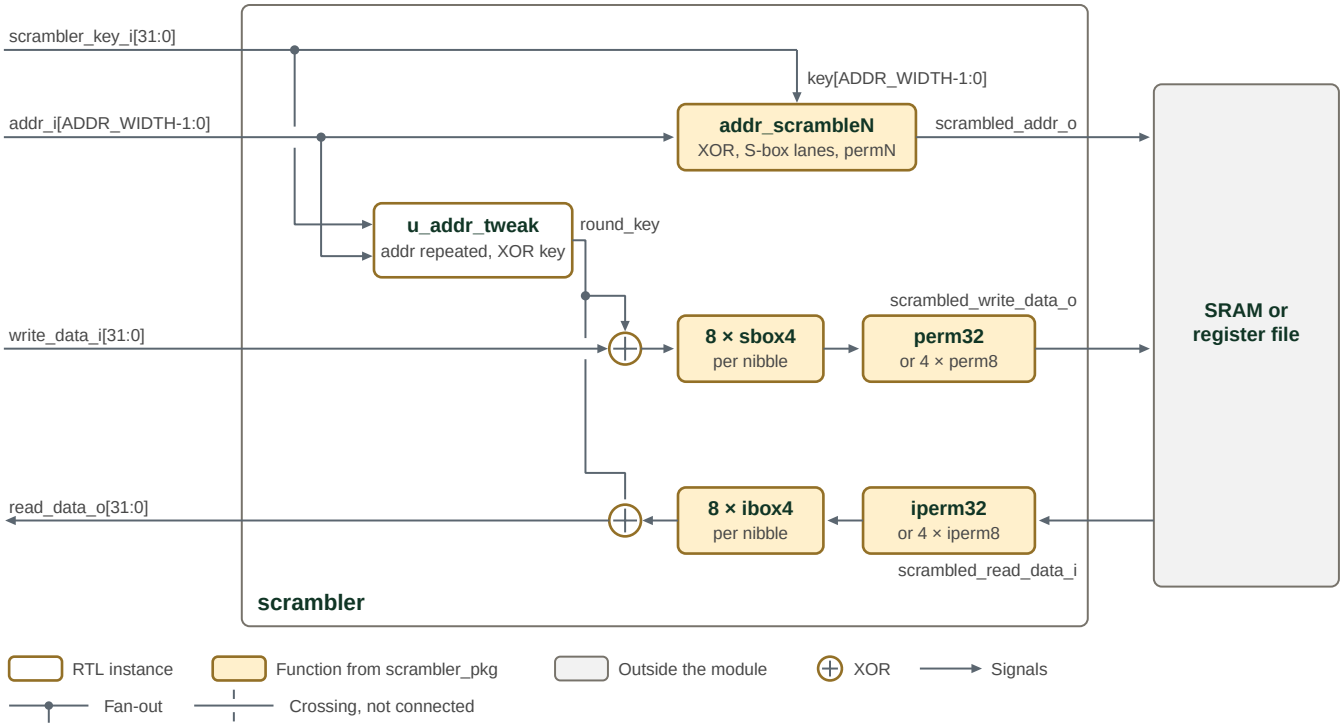


*Figure 48. SRAM scrambler datapath*

Inputs enter on the left and the memory side is on the right. The write path runs left to right along the middle row and the read path right to left along the bottom row. Both paths use the same `round_key` from `u_addr_tweak`, so the read path must be given the address the read data was fetched from. `BYTE_WISE` selects between the whole-word permutations and the per-byte permutations shown in each permutation box. The enable multiplexer and the parity logic belong to the caller, not to `scrambler`.

The Key and Policy Vault (`km_kpv`) also instantiates `scrambler_1024x32` (`u_scrambler_km`, `BYTE_WISE` = 0) in front of its key register file, keyed by the KPV's own scrambler key.

### Firmware

The KM ROM and any mutable firmware it hands over to run on the PicoRV32. The Key Manager Application ROM appendix of the Programmer's Guide is the firmware reference: initialization, the mailbox command and response protocol, interrupt handling, key management and the SRAM handover rules.

## Memory Map and Register Reference

### Memory Map

The KM CPU addresses its internal peripherals through a flat 32-bit address space. The AXI4-Lite crossbar decodes the address and routes transactions to the corresponding slave port.

Every window except OTP/eFuse spans exactly the address bits its register block decodes, so no register is reachable from a second address. The last two columns of the map state how the crossbar and the register blocks answer an access that no register or memory backs, and Decode Response Codes defines them and their codes. The KM CPU does not see these responses: an error sets `AXI_SLVERR` or `AXI_DECERR` in the KMCSR `IRQ_STATUS` register.

The OTP/eFuse window is the exception: hardware preserves its lower 12 address bits and replaces the upper bits with the integration parameter `OTP_EFUSE_REMAP_BASE` before forwarding the transaction to SEP. Only the MAP, CTRL and MMR register maps are forwarded. The Key Manager answers any other offset in the window with SLVERR, so the KM CPU cannot reach the eFuse shim control registers.

*Table 394. Key Manager CPU Memory Map*

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x00000000 | 0x00003FFF | 16 KiB | 16 KiB | Key Manager ROM | Key Manager boot ROM | -[1] | - |
| 0x00004000 | 0x00007FFF | 16 KiB | 16 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x00008000 | 0x0000FFFF | 32 KiB | 32 KiB | Key Manager SRAM | Key Manager data and executable SRAM | - | - |
| 0x00010000 | 0x0001001F | 32 B | 28 B | Key Manager Mailbox KM Side | Register interface for KM CPU to access mailbox (write to outbound FIFO, read from inbound FIFO) | - | SLVERR, 0x0 / SLVERR |
| 0x00010020 | 0x00010FFF | 4064 B | 4064 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x00011000 | 0x000113FF | 1 KiB | 1 KiB | SEP eFuse Map Address Map | SEP eFuse shadow-register pass-through. | Forwarded | - |
| 0x00011400 | 0x000114FF | 256 B | 28 B | eFuse Interface Control Address Map | SEP eFuse command and status pass-through. | Forwarded | SLVERR, 0xBADCAB1E / SLVERR |
| 0x00011500 | 0x000115FF | 256 B | 116 B | eFuse Token Registers Address Map | SEP eFuse security-token pass-through. | Forwarded | SLVERR, 0xBADCAB1E / SLVERR |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x00011600 | 0x00011FFF | 2560 B | 2560 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR[2] | - |
| 0x00012000 | 0x00013FFF | 8 KiB | 4360 B | Key and Policy Vault (KM Port) | Key entry storage and KM control registers. Access conditioned by lock_write/lock_use. | - | SLVERR, 0x0 / SLVERR |
| 0x00014000 | 0x000147FF | 2 KiB | 1216 B | Key Manager Control and Status Registers | Control and status registers for the Key Manager subsystem | SLVERR, 0x0 / SLVERR | SLVERR, 0x0 / SLVERR |
| 0x00014800 | 0x00014FFF | 2 KiB | 2 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x00015000 | 0x0001500F | 16 B | 16 B | DRBG Sampler | DRBG random data interface and configuration/status registers | - | - |
| 0x00015010 | 0x00017FFF | 12272 B | 12272 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x00018000 | 0x0001807F | 128 B | 100 B | OTBN Wrapper Key Registers | Key storage registers for the OTBN sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT OTBN core. | - | SLVERR, 0x0 / SLVERR |
| 0x00018080 | 0x00018FFF | 3968 B | 3968 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x00019000 | 0x0001907F | 128 B | 68 B | AES Wrapper Key Registers | Key storage registers for the AES sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT AES core. | - | SLVERR, 0x0 / SLVERR |
| 0x00019080 | 0x00019FFF | 3968 B | 3968 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x0001A000 | 0x0001A07F | 128 B | 68 B | KMAC Wrapper Key Registers | Key storage registers for the KMAC sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT KMAC core. | - | SLVERR, 0x0 / SLVERR |
| 0x0001A080 | 0x0001AFFF | 3968 B | 3968 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x0001B000 | 0x0001B07F | 128 B | 68 B | HMAC Wrapper Key Registers | Key storage registers for the HMAC sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT HMAC core. | - | SLVERR, 0x0 / SLVERR |
| 0x0001B080 | 0x0001BFFF | 3968 B | 3968 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x0001C000 | 0x0001C7FF | 2 KiB | 1068 B | Adams Bridge Key Sideload Registers | Single AXI4-Lite window for all KM sideload interfaces to Adams Bridge. Contains four dual-share secret-seed inputs and one shared-key output. | SLVERR, 0x0 / SLVERR | SLVERR, 0x0 / SLVERR |
| 0x0001C800 | 0x0001CFFF | 2 KiB | 2 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |

*Notes*

[1] A write completes without an error response, is dropped, and sets IRQ_STATUS.ROM_WRITE_ERR.
[2] This range is inside the OTP/eFuse window, which answers every offset outside its register maps with SLVERR and read data 0xBADCAB1E rather than DECERR.

### Detailed Register Map

### Address Summary Registers

### Key Manager Top-Level Address Map

Complete memory map for the Key Manager subsystem, including ROM, SRAM, control/status registers, mailbox, and external crypto engine ports. This address map is from the perspective of the KM CPU (PicoRV32).

| NOTE | **Register Arrays:** |
|---|---|

- **RMA_SIP_TOKEN_I[8]**: 8 registers, base 0x11500, stride 0x4
- **RMA_CHIPLET_TOKEN_I[8]**: 8 registers, base 0x11520, stride 0x4
- **SEC_DISABLE_TOKEN_I[8]**: 8 registers, base 0x11540, stride 0x4
- **KEY_ENTRY[64].WORD[16]**: 1024 registers, base 0x12000 (KEY_ENTRY x64 stride 0x40, WORD x16 stride 0x4)
- **CTRL[64]**: 64 registers, base 0x13000, stride 0x4
- **KEY_SHARE0[12]**: 12 registers, base 0x18000, stride 0x4
- **KEY_SHARE1[12]**: 12 registers, base 0x18030, stride 0x4
- **aes_wrapper_key.KEY_SHARE0[8]**: 8 registers, base 0x19000, stride 0x4
- **aes_wrapper_key.KEY_SHARE1[8]**: 8 registers, base 0x19020, stride 0x4
- **kmac_wrapper_key.KEY_SHARE0[8]**: 8 registers, base 0x1A000, stride 0x4
- **kmac_wrapper_key.KEY_SHARE1[8]**: 8 registers, base 0x1A020, stride 0x4
- **hmac_wrapper_key.KEY_SHARE0[8]**: 8 registers, base 0x1B000, stride 0x4
- **hmac_wrapper_key.KEY_SHARE1[8]**: 8 registers, base 0x1B020, stride 0x4
- **abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8]**: 8 registers, base 0x1C000, stride 0x4
- **abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8]**: 8 registers, base 0x1C020, stride 0x4
- **abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8]**: 8 registers, base 0x1C100, stride 0x4
- **abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8]**: 8 registers, base 0x1C120, stride 0x4
- **abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8]**: 8 registers, base 0x1C200, stride 0x4
- **abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8]**: 8 registers, base 0x1C220, stride 0x4
- **abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8]**: 8 registers, base 0x1C300, stride 0x4
- **abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8]**: 8 registers, base 0x1C320, stride 0x4
- **KEY[8]**: 8 registers, base 0x1C400, stride 0x4

*Table 395. Key Manager Top-Level Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x10000 | KM_WRITE_DATA | W | Write data to outbound FIFO (KM→SEP messages) |
| 0x10004 | KM_WRITE_SEPARATOR | RW | Write 1 to set message separator on next outbound (KM→SEP) write. Cleared by hardware when that write completes. |
| 0x10008 | KM_READ_DATA | R | Read data from inbound FIFO (SEP→KM messages) |
| 0x1000C | KM_STATUS | RW | FIFO status information |
| 0x10010 | KM_IRQ_STATUS | RW | Interrupt status register |
| 0x10014 | KM_IRQ_ENABLE | RW | Interrupt enable register for KM CPU |
| 0x10018 | KM_CTRL | RW | Control register for mailbox behavior configuration |
| 0x11000 | LOCKS | RW | - |
| 0x11008 | LOCKS_SPARE | RW | - |
| 0x1100C | LC_STATE | R | - |
| 0x11010 | SBOOT_DIS | R | - |
| 0x11014 | TRANSIENT_RMA_EN | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x11018 | SIP_DIS | RW | An owner's feature control vector. When set (1'b1), the corresponding feature is disabled. Layout: see the Disable Vector Format section of lifecycle_controller.adoc. |
| 0x11020 | SYS_DIS | RW | An owner's feature control vector. When set (1'b1), the corresponding feature is disabled. Layout: see the Disable Vector Format section of lifecycle_controller.adoc. |
| 0x11028 | RMA_SIP_TOKEN_DIGEST | R | - |
| 0x11048 | RMA_CHIPLET_TOKEN_DIGEST | R | - |
| 0x11068 | CLASS_KEY | R | - |
| 0x11088 | CHIPLET_PUBK_REVOKE | R | - |
| 0x1108C | BL1_VERSION | R | - |
| 0x110AC | BL2_VERSION | R | - |
| 0x110CC | CHIPLET_UID | R | - |
| 0x110EC | SIP_PUBK_HASH0 | R | - |
| 0x1110C | SIP_UID | R | - |
| 0x1112C | SYS_PUBK_HASH | R | - |
| 0x1114C | SYS_UID | R | - |
| 0x1116C | STATUS_RPT | R | - |
| 0x11170 | ROM_CTL | R | - |
| 0x11174 | SYSCLK_FREQ_MHZ | R | - |
| 0x11178 | CHIPLET_PUBK_HASH0 | R | - |
| 0x11198 | CHIPLET_PUBK_HASH1 | R | - |
| 0x111B8 | REQUIRED_SIGNERS | RW | - |
| 0x111BC | REQUIRED_ALGS | RW | - |
| 0x111C0 | CHIPLET_PUBK_PQC_HASH0 | R | - |
| 0x111E0 | CHIPLET_PUBK_PQC_HASH1 | R | - |
| 0x11200 | SIP_PUBK_PQC_HASH0 | R | - |
| 0x11220 | SYS_PUBK_PQC_HASH | R | - |
| 0x11240 | SIP_PUBK_HASH1 | R | - |
| 0x11260 | SIP_PUBK_PQC_HASH1 | R | - |
| 0x11280 | SEP_CHIPLET_ID | R | - |
| 0x112A0 | SEP_SIP_ID | R | - |
| 0x112C0 | SEP_SYS_ID | R | - |
| 0x112E0 | spare0 | R | - |
| 0x11300 | spare1 | R | - |
| 0x11320 | spare2 | R | - |
| 0x11340 | spare3 | R | - |
| 0x11360 | spare4 | R | - |
| 0x11380 | spare5 | R | - |
| 0x113A0 | spare6 | R | - |
| 0x113C0 | spare7 | R | - |
| 0x113E0 | spare8 | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x11400 | EFUSE_INTERFACE_CTRL_STATUS | RW | eFuse interface status and error-clear register. Reports the sense-done flag and request/address error conditions, and provides write-1 strobes to clear those errors. |
| 0x11404 | EFUSE_PROGRAM_CTRL | RW | eFuse program control and status. Configures the program address and data, triggers a program (optionally with read-back), and reports program progress and completion status. |
| 0x11408 | EFUSE_READ_CTRL | RW | eFuse read control and status. Configures the read address, triggers a read, and reports read progress and completion status. |
| 0x1140C | EFUSE_PROGRAM_INTERFACE_READ_DATA | R | Read data captured through the program interface. Hardware debug only; valid only while the program interface state machine is in its capture-data state. |
| 0x11410 | EFUSE_READ_INTERFACE_READ_DATA | R | Read data returned through the eFuse read interface. |
| 0x11414 | EFUSE_READ_REQ_TIMEOUT | RW | Read interface timeout configuration. Sets the number of cycles the read interface waits for a SHIM response and enables the read timeout. |
| 0x11418 | EFUSE_PROGRAM_REQ_TIMEOUT | RW | Program interface timeout configuration. Sets the number of cycles the program interface waits for a SHIM response and enables the program timeout. |
| 0x11500 - 0x1151C | RMA_SIP_TOKEN_I[8] | RW | Software-writeable RMA token word. The assembled token is hashed and compared against the fused value to authorize lifecycle (LC) state transitions. |
| 0x11520 - 0x1153C | RMA_CHIPLET_TOKEN_I[8] | RW | Software-writeable RMA token word. The assembled token is hashed and compared against the fused value to authorize lifecycle (LC) state transitions. |
| 0x11540 - 0x1155C | SEC_DISABLE_TOKEN_I[8] | RW | Software-writeable security-disable token word. The assembled token is hashed and compared against the post-silicon injected value to authorize security_disable. |
| 0x11560 | TOKEN_EOP | W | Token end-of-packet control. Writing the per-token go bit signals that all token words have been written and triggers a pulse to begin token hashing. |
| 0x11564 | RMA_SIP_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x11568 | RMA_CHIPLET_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x1156C | SEC_DISABLE_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x11570 | TOKEN_MATCH_FAULT | R | Token comparator redundancy fault status. The token match logic is triple redundant and is not a majority vote: all three comparators must agree and each must drive a legal differential pair. A bit sets when the comparators for that token disagree, which may indicate a fault injection or FIB attack. The corresponding token_match_status then reads the error code 6'b111111 and an interrupt is raised to the SEP CPU. Bits are sticky until reset. |
| 0x12000 - 0x12FFC | KEY_ENTRY[64].WORD[16] | RW | One 32-bit word of a key entry (512-bit key = 16 words). No reset; powers up random. |
| 0x13000 - 0x130FC | CTRL[64] | RW | Per-slot control: lock bits (W1S) and the erase trigger |
| 0x13100 | KPV_SCRAMBLER_KEY | RW | 32-bit key for KPV key entry scrambling. No reset; powers up random. Locked when KPV_SCRAMBLER_CTRL.LOCK=1. |
| 0x13104 | KPV_SCRAMBLER_CTRL | RW | KPV scrambler enable and lock. Reset to 0 on KM reset. LOCK is write-one-only. |
| 0x14000 | VERSION | R | Hardware version register (semantic versioning: major.minor.patch) |
| 0x14004 | CTRL | R | Control register (reserved for future use) |
| 0x14008 | SOFT_RST_CODE | RW | Software reset code register. Write 0x53525354 ('SRST') to trigger soft reset. Any other value has no effect. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1400C | kmcsr.IRQ_STATUS | RW | Interrupt status register. Sticky bits cleared by writing 1. |
| 0x14010 | kmcsr.IRQ_ENABLE | RW | Interrupt enable/mask register |
| 0x14014 | SCRAMBLER_KEY | RW | 32-bit scrambler key for SRAM address/data scrambling. When locked: writes ignored, reads return 0. No reset; powers up random. |
| 0x14018 | SCRAMBLER_CTRL | RW | Scrambler control register |
| 0x1401C | SRAM_LOCK | RW | SRAM write-lock bits. Bit[i]=1 locks region i (1 KB each, 32 regions). Write-1-only: writing 1 sets the bit, writing 0 has no effect. Cleared by warm or cold reset (warm reset domain); ROM re-applies the lock policy on every boot before handover to mutable firmware. |
| 0x14020 | IRQ_SET | W | Interrupt set register. Write 1 to trigger corresponding interrupt (for ISR testing). Write-only, reads return 0. |
| 0x14024 | SRAM_WRITE_LOCK_VIOLATION | RW | Which SRAM region(s) had write attempts while locked. Bit[i]=region i. Sticky, write 1 to clear each bit. |
| 0x14028 | RECOVERABLE_ERR | RW | Recoverable error status. Firmware writes 1 to set (after recovering in ISR), writes 0 to clear (e.g. when SEP requests clear). Drives recoverable error event output. |
| 0x1402C | BOOT_STATUS | RW | Boot status. Firmware writes 1 to COLD_BOOT_DONE after cold-boot init completes; bit remains set until next cold reset. |
| 0x14030 | OTP_LIFE_CYCLE | R | Life cycle state. 8-bit (4-bit value differentially encoded). Read-through from OTP port; no reset. |
| 0x14034 | OTP_DEMOTION_STATE | R | Demotion state. 2-bit, differentially encoded. Read-through from OTP port; no reset. |
| 0x14038 | SRAM_EXEC_MODE | RW | Execute-permission whitelist mode. Write-1-only (woset); cleared by warm or cold reset. 0=ROM executable; 1=write-locked SRAM regions executable, and the ROM lockout is armed. Set by ROM stack-less handover immediately before jumping to mutable firmware. |
| 0x140B8 | IRQ_ENTRY_ADDR | RW | Effective IRQ entry address for subsequent interrupts. Writable while unlocked; any 32-bit value accepted. |
| 0x140BC | IRQ_ENTRY_LOCK | RW | Write-one-only lock for IRQ_ENTRY_ADDR. Separate readable register; cleared only on KM reset. |
| 0x14100 | VUART_TX | RW | Virtual UART transmit register. Write a byte to send to testbench. DATA_VALID is set on write and self-clears. |
| 0x14104 | VUART_RX | R | Virtual UART receive register. Testbench writes bytes here for firmware to read. |
| 0x14108 | VUART_STATUS | R | Virtual UART status register for flow control |
| 0x14110 | TB_RESULT | RW | Test result register. Firmware writes 0=fail, 1=pass. |
| 0x14114 | TB_SIGNATURE | RW | Test completion signature. Firmware writes 0x600D600D (pass) or 0xBADBADBA (fail). |
| 0x14118 | TB_ERRCODE | RW | Test error code register. Firmware writes optional error code for debugging. |
| 0x1411C | TB_SUBTEST | RW | Current subtest number register. Firmware increments for each subtest. |
| 0x14120 | TB_CMD | RW | Testbench command register. Firmware writes command, testbench reads and clears. |
| 0x14124 | TB_CMD_ARG | RW | Testbench command argument register. Firmware writes optional argument. |
| 0x14128 | TB_CMD_STATUS | RW | Testbench command status register. Testbench writes status, firmware reads. |
| 0x1412C | TB_CMD_RESULT | R | Testbench command result register. Testbench writes result value. |
| 0x141FC | DEBUG | R | Debug register with known constant value for verification |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x14200 | OTP_CHIPLET_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14204 | OTP_CHIPLET_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14208 | OTP_CHIPLET_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1420C | OTP_CHIPLET_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14210 | OTP_CHIPLET_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14214 | OTP_CHIPLET_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14218 | OTP_CHIPLET_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1421C | OTP_CHIPLET_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14220 | OTP_CHIPLET_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14224 | OTP_CHIPLET_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14228 | OTP_CHIPLET_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1422C | OTP_CHIPLET_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14230 | OTP_CHIPLET_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14234 | OTP_CHIPLET_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14238 | OTP_CHIPLET_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1423C | OTP_CHIPLET_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14240 | OTP_SIP_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14244 | OTP_SIP_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x14248 | OTP_SIP_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1424C | OTP_SIP_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14250 | OTP_SIP_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14254 | OTP_SIP_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14258 | OTP_SIP_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1425C | OTP_SIP_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14260 | OTP_SIP_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14264 | OTP_SIP_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14268 | OTP_SIP_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1426C | OTP_SIP_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14270 | OTP_SIP_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14274 | OTP_SIP_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14278 | OTP_SIP_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1427C | OTP_SIP_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14280 | OTP_SYS_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14284 | OTP_SYS_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14288 | OTP_SYS_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1428C | OTP_SYS_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x14290 | OTP_SYS_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14294 | OTP_SYS_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14298 | OTP_SYS_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1429C | OTP_SYS_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142A0 | OTP_SYS_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142A4 | OTP_SYS_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142A8 | OTP_SYS_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142AC | OTP_SYS_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142B0 | OTP_SYS_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142B4 | OTP_SYS_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142B8 | OTP_SYS_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142BC | OTP_SYS_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142C0 | OTP_CLASS_KEY_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142C4 | OTP_CLASS_KEY_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142C8 | OTP_CLASS_KEY_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142CC | OTP_CLASS_KEY_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142D0 | OTP_CLASS_KEY_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142D4 | OTP_CLASS_KEY_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x142D8 | OTP_CLASS_KEY_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142DC | OTP_CLASS_KEY_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142E0 | OTP_CLASS_KEY_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142E4 | OTP_CLASS_KEY_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142E8 | OTP_CLASS_KEY_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142EC | OTP_CLASS_KEY_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142F0 | OTP_CLASS_KEY_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142F4 | OTP_CLASS_KEY_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142F8 | OTP_CLASS_KEY_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x142FC | OTP_CLASS_KEY_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14300 | OTP_READ_LOCK | RW | OTP field warm-reset-domain read-lock bits. Write 1 to set; writing 0 has no effect. Locked reads of the corresponding OTP field return zero (effective lock = OTP_READ_LOCK \| OTP_READ_LOCK_COLD). Warm-reset domain: cleared by warm or cold reset. |
| 0x14304 | OTP_CHANGE_STATUS | RW | OTP field change status. A bit is set when the corresponding field's decoded value changes. Sticky, write 1 to clear. Valid after IRQ_STATUS.OTP_CHANGE fires. |
| 0x14308 | OTP_READ_LOCK_COLD | RW | OTP field cold-reset-domain read-lock bits. Write 1 to set; writing 0 has no effect. Locked reads of the corresponding OTP field return zero (effective lock = OTP_READ_LOCK \| OTP_READ_LOCK_COLD). Cold-reset domain: survives warm reset. |
| 0x14400 | OTP_SEP_CHIPLET_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14404 | OTP_SEP_CHIPLET_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14408 | OTP_SEP_CHIPLET_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1440C | OTP_SEP_CHIPLET_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x14410 | OTP_SEP_CHIPLET_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14414 | OTP_SEP_CHIPLET_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14418 | OTP_SEP_CHIPLET_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1441C | OTP_SEP_CHIPLET_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14420 | OTP_SEP_CHIPLET_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14424 | OTP_SEP_CHIPLET_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14428 | OTP_SEP_CHIPLET_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1442C | OTP_SEP_CHIPLET_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14430 | OTP_SEP_CHIPLET_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14434 | OTP_SEP_CHIPLET_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14438 | OTP_SEP_CHIPLET_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1443C | OTP_SEP_CHIPLET_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14440 | OTP_SEP_SIP_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14444 | OTP_SEP_SIP_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14448 | OTP_SEP_SIP_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1444C | OTP_SEP_SIP_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14450 | OTP_SEP_SIP_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14454 | OTP_SEP_SIP_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x14458 | OTP_SEP_SIP_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1445C | OTP_SEP_SIP_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14460 | OTP_SEP_SIP_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14464 | OTP_SEP_SIP_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14468 | OTP_SEP_SIP_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1446C | OTP_SEP_SIP_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14470 | OTP_SEP_SIP_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14474 | OTP_SEP_SIP_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14478 | OTP_SEP_SIP_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1447C | OTP_SEP_SIP_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14480 | OTP_SEP_SYS_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14484 | OTP_SEP_SYS_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14488 | OTP_SEP_SYS_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1448C | OTP_SEP_SYS_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14490 | OTP_SEP_SYS_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14494 | OTP_SEP_SYS_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x14498 | OTP_SEP_SYS_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x1449C | OTP_SEP_SYS_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x144A0 | OTP_SEP_SYS_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144A4 | OTP_SEP_SYS_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144A8 | OTP_SEP_SYS_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144AC | OTP_SEP_SYS_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144B0 | OTP_SEP_SYS_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144B4 | OTP_SEP_SYS_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144B8 | OTP_SEP_SYS_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x144BC | OTP_SEP_SYS_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x15000 | DATA | R | One word of random data. Read triggers request or returns prefetch. Writes return SLVERR. No reset. |
| 0x15004 | CFG | RW | Configuration: PREFETCH enable, TIMEOUT cycles for active CPU read. |
| 0x15008 | STATUS | RW | Status: DRBG_READY, PREFETCHED, TIMEOUT_ERR (W1C), STREAM_ERR (W1C), COUNT_BAD, COUNT_GOOD. |
| 0x1500C | PREFETCH_DATA | R | Prefetched word (read-only, for debugging). Cleared when PREFETCH=0. No reset; power-up undefined. |
| 0x18000 - 0x1802C | KEY_SHARE0[12] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x18030 - 0x1805C | KEY_SHARE1[12] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x18060 | otbn_wrapper_key.KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the OTBN core. |
| 0x19000 - 0x1901C | aes_wrapper_key.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x19020 - 0x1903C | aes_wrapper_key.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x19040 | aes_wrapper_key.KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the AES core. |
| 0x1A000 - 0x1A01C | kmac_wrapper_key.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1A020 - 0x1A03C | kmac_wrapper_key.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1A040 | kmac_wrapper_key.KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the KMAC core. |
| 0x1B000 - 0x1B01C | hmac_wrapper_key.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1B020 - 0x1B03C | hmac_wrapper_key.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1B040 | hmac_wrapper_key.KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the HMAC core. |
| 0x1C000 - 0x1C01C | abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C020 - 0x1C03C | abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C040 | abr_wrapper_key.MLDSA_SEED.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x1C100 - 0x1C11C | abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C120 - 0x1C13C | abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C140 | abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x1C200 - 0x1C21C | abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C220 - 0x1C23C | abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C240 | abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x1C300 - 0x1C31C | abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C320 - 0x1C33C | abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x1C340 | abr_wrapper_key.MLKEM_MSG.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x1C400 - 0x1C41C | KEY[8] | R | One 32-bit word of ML-KEM shared key output. Written by hardware when key is ready; read by KM firmware to consume. Hardware clears all words to 0 when KEY_CTRL.KEY_VALID is cleared (via hwclr). |
| 0x1C420 | abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL | RW | Shared-key control. HW sets KEY_VALID when a new key is ready. Firmware writes 0 to clear (consume and zeroize). |
| 0x1C424 | abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_STATUS | RW | Interrupt status for ML-KEM shared key. Sticky bit set by hardware when a new shared key becomes valid; cleared by writing 1. |
| 0x1C428 | abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_ENABLE | RW | Interrupt enable for ML-KEM shared key. Mask for the PicoRV32 IRQ vector bit driven by the shared-key-valid event. |

### KM_WRITE_DATA

*Table 396. key_manager.KM_WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | 0x0 | 32-bit data word to write to outbound FIFO |

### KM_WRITE_SEPARATOR

*Table 397. key_manager.KM_WRITE_SEPARATOR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `set` | RW | 0x0 | Write 1 to set message separator on next outbound write. Cleared by hardware when that write completes. |

### KM_READ_DATA

*Table 398. key_manager.KM_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | 0x0 | 32-bit data word read from inbound FIFO |

### KM_STATUS

*Table 399. key_manager.KM_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:26 | `rsvd` | R | 0x0 | Reserved |
| 25 | `outbound_separator` | R | 0x0 | Message separator: 1 if the last word read from outbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 24 | `inbound_separator` | R | 0x0 | Message separator: 1 if the last word read from inbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 23 | `outbound_underflow` | RW1C | 0x0 | Outbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |
| 22 | `inbound_underflow` | RW1C | 0x0 | Inbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |
| 21 | `outbound_overflow` | RW1C | 0x0 | Outbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 20 | `inbound_overflow` | RW1C | 0x0 | Inbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 19:12 | `outbound_depth` | R | 0x0 | Outbound FIFO fill level (number of words in FIFO) |
| 11:4 | `inbound_depth` | R | 0x0 | Inbound FIFO fill level (number of words in FIFO) |
| 3 | `outbound_full` | R | 0x0 | Outbound FIFO is full (cannot write more data) |
| 2 | `outbound_empty` | R | 0x1 | Outbound FIFO is empty (no data available for SEP to read) |
| 1 | `inbound_full` | R | 0x0 | Inbound FIFO is full |
| 0 | `inbound_empty` | R | 0x1 | Inbound FIFO is empty (no data available for KM to read) |

### KM_IRQ_STATUS

*Table 400. key_manager.KM_IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_sep` | RW1C | 0x0 | SEP performed a mailbox flush. Sticky, write 1 to clear. |
| 3 | `inbound_underflow` | RW1C | 0x0 | Inbound FIFO underflow detected. Sticky, write 1 to clear. |
| 2 | `outbound_overflow` | RW1C | 0x0 | Outbound FIFO overflow detected. Sticky, write 1 to clear. |
| 1 | `outbound_write_space_avail` | R | 0x0 | Outbound FIFO has space available for KM to write (level-sensitive). 1 = space available, 0 = full. |
| 0 | `inbound_read_data_avail` | R | 0x0 | Inbound FIFO has data available for KM to read (level-sensitive). 1 = data available, 0 = empty. |

### KM_IRQ_ENABLE

*Table 401. key_manager.KM_IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_sep_en` | RW | 0x0 | Enable interrupt to KM when SEP performs a mailbox flush |
| 3 | `inbound_underflow_en` | RW | 0x0 | Enable interrupt to KM when inbound FIFO underflow is detected |
| 2 | `outbound_overflow_en` | RW | 0x0 | Enable interrupt to KM when outbound FIFO overflow is detected |
| 1 | `outbound_write_space_avail_en` | RW | 0x0 | Enable interrupt to KM when outbound FIFO has space available (KM→SEP messages) |
| 0 | `inbound_read_data_avail_en` | RW | 0x0 | Enable interrupt to KM when inbound FIFO has data available (SEP→KM messages) |

### KM_CTRL

*Table 402. key_manager.KM_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `rsvd` | R | 0x0 | Reserved |
| 2 | `flush` | RW | 0x0 | Write 1 to flush all mailbox FIFOs (inbound and outbound). Cleared by hardware when flush completes. |
| 1 | `inbound_underflow_resp` | RW | 0x0 | Response type for inbound FIFO underflow: 0=SLVERR (default), 1=OKAY |
| 0 | `outbound_overflow_resp` | RW | 0x0 | Response type for outbound FIFO overflow: 0=SLVERR (default), 1=OKAY |

### LOCKS

*Table 403. key_manager.LOCKS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `SEP_SYS_ID_READ_LOCK` | RW1S | 0x0 | sep_sys_id_read_lock (slot 31): when set, SEP_SYS_ID is not read-able. |
| 62 | `SEP_SYS_ID_WRITE_LOCK` | RW1S | 0x0 | sep_sys_id_write_lock (slot 31): when set, SEP_SYS_ID is not write-able. |
| 61 | `SEP_SIP_ID_READ_LOCK` | RW1S | 0x0 | sep_sip_id_read_lock (slot 30): when set, SEP_SIP_ID is not read-able. |
| 60 | `SEP_SIP_ID_WRITE_LOCK` | RW1S | 0x0 | sep_sip_id_write_lock (slot 30): when set, SEP_SIP_ID is not write-able. |
| 59 | `SEP_CHIPLET_ID_READ_LOCK` | RW1S | 0x0 | sep_chiplet_id_read_lock (slot 29): when set, SEP_CHIPLET_ID is not read-able. |
| 58 | `SEP_CHIPLET_ID_WRITE_LOCK` | RW1S | 0x0 | sep_chiplet_id_write_lock (slot 29): when set, SEP_CHIPLET_ID is not write-able. |
| 57 | `SIP_PUBK_PQC_HASH1_READ_LOCK` | RW1S | 0x0 | sip_pubk_pqc_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 56 | `SIP_PUBK_PQC_HASH1_WRITE_LOCK` | RW1S | 0x0 | sip_pubk_pqc_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 55 | `SIP_PUBK_HASH1_READ_LOCK` | RW1S | 0x0 | sip_pubk_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 54 | `SIP_PUBK_HASH1_WRITE_LOCK` | RW1S | 0x0 | sip_pubk_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 53 | `SYS_PUBK_PQC_HASH_READ_LOCK` | RW1S | 0x0 | sys_pubk_pqc_hash_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 52 | `SYS_PUBK_PQC_HASH_WRITE_LOCK` | RW1S | 0x0 | sys_pubk_pqc_hash_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 51 | SIP_PUBK_PQC_HASH0_READ_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 50 | SIP_PUBK_PQC_HASH0_WRITE_LOCK | RW1S | 0x0 | sip_pubk_pqc_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 49 | CHIPLET_PUBK_PQC_HASH1_READ_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 48 | CHIPLET_PUBK_PQC_HASH1_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 47 | CHIPLET_PUBK_PQC_HASH0_READ_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 46 | CHIPLET_PUBK_PQC_HASH0_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_pqc_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 45 | REQUIRED_ALGS_READ_LOCK | RW1S | 0x0 | required_algs_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 44 | REQUIRED_ALGS_WRITE_LOCK | RW1S | 0x0 | required_algs_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 43 | REQUIRED_SIGNERS_READ_LOCK | RW1S | 0x0 | required_signers_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 42 | REQUIRED_SIGNERS_WRITE_LOCK | RW1S | 0x0 | required_signers_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 41 | CHIPLET_PUBK_HASH1_READ_LOCK | RW1S | 0x0 | chiplet_pubk_hash1_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 40 | CHIPLET_PUBK_HASH1_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_hash1_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 39 | CHIPLET_PUBK_HASH0_READ_LOCK | RW1S | 0x0 | chiplet_pubk_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 38 | CHIPLET_PUBK_HASH0_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 37 | SYSCLK_FREQ_MHZ_READ_LOCK | RW1S | 0x0 | sysclk_freq_mhz_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 36 | SYSCLK_FREQ_MHZ_WRITE_LOCK | RW1S | 0x0 | sysclk_freq_mhz_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 35 | ROM_CTL_READ_LOCK | RW1S | 0x0 | rom_ctl_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 34 | ROM_CTL_WRITE_LOCK | RW1S | 0x0 | rom_ctl_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 33 | STATUS_RPT_READ_LOCK | RW1S | 0x0 | status_rpt_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 32 | STATUS_RPT_WRITE_LOCK | RW1S | 0x0 | status_rpt_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 31 | SYS_UID_READ_LOCK | RW1S | 0x0 | sys_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 30 | SYS_UID_WRITE_LOCK | RW1S | 0x0 | sys_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 29 | SYS_PUBK_HASH_READ_LOCK | RW1S | 0x0 | sys_pubk_hash_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 28 | SYS_PUBK_HASH_WRITE_LOCK | RW1S | 0x0 | sys_pubk_hash_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 27 | SIP_UID_READ_LOCK | RW1S | 0x0 | sip_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 26 | SIP_UID_WRITE_LOCK | RW1S | 0x0 | sip_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 25 | SIP_PUBK_HASH0_READ_LOCK | RW1S | 0x0 | sip_pubk_hash0_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 24 | SIP_PUBK_HASH0_WRITE_LOCK | RW1S | 0x0 | sip_pubk_hash0_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 23 | CHIPLET_UID_READ_LOCK | RW1S | 0x0 | chiplet_uid_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 22 | CHIPLET_UID_WRITE_LOCK | RW1S | 0x0 | chiplet_uid_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 21 | BL2_VERSION_READ_LOCK | RW1S | 0x0 | bl2_version_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 20 | BL2_VERSION_WRITE_LOCK | RW1S | 0x0 | bl2_version_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 19 | BL1_VERSION_READ_LOCK | RW1S | 0x0 | bl1_version_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 18 | BL1_VERSION_WRITE_LOCK | RW1S | 0x0 | bl1_version_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 17 | CHIPLET_PUBK_REVOKE_READ_LOCK | RW1S | 0x0 | chiplet_pubk_revoke_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 16 | CHIPLET_PUBK_REVOKE_WRITE_LOCK | RW1S | 0x0 | chiplet_pubk_revoke_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 15 | CLASS_KEY_READ_LOCK | RW1S | 0x0 | class_key_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 14 | CLASS_KEY_WRITE_LOCK | RW1S | 0x0 | class_key_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 13 | RMA_CHIPLET_TOKEN_DIGEST_READ_LOCK | RW1S | 0x0 | rma_chiplet_token_digest_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 12 | RMA_CHIPLET_TOKEN_DIGEST_WRITE_LOCK | RW1S | 0x0 | rma_chiplet_token_digest_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 11 | RMA_SIP_TOKEN_DIGEST_READ_LOCK | RW1S | 0x0 | rma_sip_token_digest_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 10 | RMA_SIP_TOKEN_DIGEST_WRITE_LOCK | RW1S | 0x0 | rma_sip_token_digest_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 9 | SYS_DIS_READ_LOCK | RW1S | 0x0 | sys_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 8 | SYS_DIS_WRITE_LOCK | RW1S | 0x0 | sys_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 7 | SIP_DIS_READ_LOCK | RW1S | 0x0 | sip_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 6 | SIP_DIS_WRITE_LOCK | RW1S | 0x0 | sip_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 5 | TRANSIENT_RMA_EN_READ_LOCK | RW1S | 0x0 | TRANSIENT_RMA_EN_READ_LOCK, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 4 | TRANSIENT_RMA_EN_WRITE_LOCK | RW1S | 0x0 | TRANSIENT_RMA_EN_WRITE_LOCK, when set, this field is not write-able (neither the eFuse nor the shadow register) |
| 3 | SBOOT_DIS_READ_LOCK | RW1S | 0x0 | sboot_dis_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 2 | SBOOT_DIS_WRITE_LOCK | RW1S | 0x0 | sboot_dis_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | LC_STATE_READ_LOCK | RW1S | 0x0 | lc_state_read_lock, when set, this field is not read-able (neither the eFuse nor the shadow register) |
| 0 | LC_STATE_WRITE_LOCK | RW1S | 0x0 | lc_state_write_lock, when set, this field is not write-able (neither the eFuse nor the shadow register) |

### LOCKS_SPARE

*Table 404. key_manager.LOCKS_SPARE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:18 | SPARE_LOCK_RSVD | RW1S | 0x0 | Unassigned lock slots 41-47. Reserved for future field lock assignments. |
| 17 | SPARE8_READ_LOCK | RW1S | 0x0 | spare8_read_lock (slot 40): when set, spare8 is not read-able. |
| 16 | SPARE8_WRITE_LOCK | RW1S | 0x0 | spare8_write_lock (slot 40): when set, spare8 is not write-able. |
| 15 | SPARE7_READ_LOCK | RW1S | 0x0 | spare7_read_lock (slot 39): when set, spare7 is not read-able. |
| 14 | SPARE7_WRITE_LOCK | RW1S | 0x0 | spare7_write_lock (slot 39): when set, spare7 is not write-able. |
| 13 | SPARE6_READ_LOCK | RW1S | 0x0 | spare6_read_lock (slot 38): when set, spare6 is not read-able. |
| 12 | SPARE6_WRITE_LOCK | RW1S | 0x0 | spare6_write_lock (slot 38): when set, spare6 is not write-able. |
| 11 | SPARE5_READ_LOCK | RW1S | 0x0 | spare5_read_lock (slot 37): when set, spare5 is not read-able. |
| 10 | SPARE5_WRITE_LOCK | RW1S | 0x0 | spare5_write_lock (slot 37): when set, spare5 is not write-able. |
| 9 | SPARE4_READ_LOCK | RW1S | 0x0 | spare4_read_lock (slot 36): when set, spare4 is not read-able. |
| 8 | SPARE4_WRITE_LOCK | RW1S | 0x0 | spare4_write_lock (slot 36): when set, spare4 is not write-able. |
| 7 | SPARE3_READ_LOCK | RW1S | 0x0 | spare3_read_lock (slot 35): when set, spare3 is not read-able. |
| 6 | SPARE3_WRITE_LOCK | RW1S | 0x0 | spare3_write_lock (slot 35): when set, spare3 is not write-able. |
| 5 | SPARE2_READ_LOCK | RW1S | 0x0 | spare2_read_lock (slot 34): when set, spare2 is not read-able. |
| 4 | SPARE2_WRITE_LOCK | RW1S | 0x0 | spare2_write_lock (slot 34): when set, spare2 is not write-able. |
| 3 | SPARE1_READ_LOCK | RW1S | 0x0 | spare1_read_lock (slot 33): when set, spare1 is not read-able. |
| 2 | SPARE1_WRITE_LOCK | RW1S | 0x0 | spare1_write_lock (slot 33): when set, spare1 is not write-able. |
| 1 | SPARE0_READ_LOCK | RW1S | 0x0 | spare0_read_lock (slot 32): when set, spare0 is not read-able. |
| 0 | SPARE0_WRITE_LOCK | RW1S | 0x0 | spare0_write_lock (slot 32): when set, spare0 is not write-able. |

### LC_STATE

*Table 405. key_manager.LC_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | rsvd | R | 0x0 | Unused |
| 7:0 | lc_state | R | 0xF0 | Life cycle manufacturing state, differentially encoded: {~raw[3:0], raw[3:0]} |

### SBOOT_DIS

*Table 406. key_manager.SBOOT_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | rsvd | R | 0x0 | Unused |
| 0 | disable_secure_boot | R | 0x0 | Chicken bit to disable secure boot in SEP ROM. |

### TRANSIENT_RMA_EN

*Table 407. key_manager.TRANSIENT_RMA_EN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Unused |
| 0 | `transient_rma_en` | R | 0x0 | When Transient RMA EN is 1, matching on corresponding *_Tokens will automatically transfer the chip to the corresponding RMA state by setting the LC_STATE reg |

### SIP_DIS

*Table 408. key_manager.SIP_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:48 | `func_reserved` | RW1S | 0x0 | Reserved for adopter-defined functional feature gating (bits 63:48). |
| 47:25 | `debug_reserved_dbg2` | RW1S | 0x0 | Reserved for adopter-defined inter-chiplet-bypass (DBG_2) debug scan chains (bits 47:25). |
| 24 | `sip_debug` | RW1S | 0x0 | SIP_DBG (bit 24): Coarse-grained debug enable for the SiP scope: the inter-chiplet debug-bypass gate that admits external (PTAP) debug traffic into this chiplet and forwards it onward to the next chiplet via the outbound JTAG PTAP. 1'b1: disabled. 1'b0: enabled. |
| 23:4 | `debug_reserved_dbg1` | RW1S | 0x0 | Reserved for adopter-defined local (DBG_1) debug scan chains (bits 23:4). |
| 3 | `smc_fuse_dbg` | RW1S | 0x0 | SMC_FUSE_DBG (bit 3): 1'b1: the DFT-inserted access path into the SMC OTP is disabled. 1'b0: enabled. Produced by SEP-LCC feat_ctrl_o, exported on the SEP boundary, and consumed by SMC to gate its own inserted fuse path. |
| 2 | `sep_fuse_dbg` | RW1S | 0x0 | SEP_FUSE_DBG (bit 2): 1'b1: the DFT-inserted access path into the SEP OTP is disabled. 1'b0: enabled. Also covers the OTP IP-vendor test and calibration interface, which is reachable only over that inserted path. The functional JTAG2AXIL path to the OTP is always available and is not gated by this bit. |
| 1 | `chiplet_dbg` | RW1S | 0x0 | CHIPLET_DBG (bit 1): Coarse-grained debug enable for the chiplet scope. 1'b1: chiplet-scope debug/scan-dump access disabled. 1'b0: enabled. Never acts alone: an external debugger reaches chiplet-internal resources only when SIP_DBG is open as well. Contributes to the reachability of the crypto/KM secure chain but never authorizes a dump from it. |
| 0 | `sep_debug` | RW1S | 0x0 | SEP_DBG (bit 0): Coarse-grained debug enable for the SEP scope. 1'b1: SEP-scope debug/scan-dump access disabled and external inbound traffic is filtered. 1'b0: enabled. A local gate only: reaching SEP internal state from an external debugger additionally requires SIP_DBG and CHIPLET_DBG. |

### SYS_DIS

*Table 409. key_manager.SYS_DIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:48 | `func_reserved` | RW1S | 0x0 | Reserved for adopter-defined functional feature gating (bits 63:48). |
| 47:25 | `debug_reserved_dbg2` | RW1S | 0x0 | Reserved for adopter-defined inter-chiplet-bypass (DBG_2) debug scan chains (bits 47:25). |
| 24 | `sip_debug` | RW1S | 0x0 | SIP_DBG (bit 24): Coarse-grained debug enable for the SiP scope: the inter-chiplet debug-bypass gate that admits external (PTAP) debug traffic into this chiplet and forwards it onward to the next chiplet via the outbound JTAG PTAP. 1'b1: disabled. 1'b0: enabled. |
| 23:4 | `debug_reserved_dbg1` | RW1S | 0x0 | Reserved for adopter-defined local (DBG_1) debug scan chains (bits 23:4). |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3 | `smc_fuse_dbg` | RW1S | 0x0 | SMC_FUSE_DBG (bit 3): 1'b1: the DFT-inserted access path into the SMC OTP is disabled. 1'b0: enabled. Produced by SEP-LCC feat_ctrl_o, exported on the SEP boundary, and consumed by SMC to gate its own inserted fuse path. |
| 2 | `sep_fuse_dbg` | RW1S | 0x0 | SEP_FUSE_DBG (bit 2): 1'b1: the DFT-inserted access path into the SEP OTP is disabled. 1'b0: enabled. Also covers the OTP IP-vendor test and calibration interface, which is reachable only over that inserted path. The functional JTAG2AXIL path to the OTP is always available and is not gated by this bit. |
| 1 | `chiplet_dbg` | RW1S | 0x0 | CHIPLET_DBG (bit 1): Coarse-grained debug enable for the chiplet scope. 1'b1: chiplet-scope debug/scan-dump access disabled. 1'b0: enabled. Never acts alone: an external debugger reaches chiplet-internal resources only when SIP_DBG is open as well. Contributes to the reachability of the crypto/KM secure chain but never authorizes a dump from it. |
| 0 | `sep_debug` | RW1S | 0x0 | SEP_DBG (bit 0): Coarse-grained debug enable for the SEP scope. 1'b1: SEP-scope debug/scan-dump access disabled and external inbound traffic is filtered. 1'b0: enabled. A local gate only: reaching SEP internal state from an external debugger additionally requires SIP_DBG and CHIPLET_DBG. |

### RMA_SIP_TOKEN_DIGEST

*Table 410. key_manager.RMA_SIP_TOKEN_DIGEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `token_digest` | R | 0x0 | Hash digest of life cycle secret that allows entering RMA_SIP state / Note this is a 256 bit field |

### RMA_CHIPLET_TOKEN_DIGEST

*Table 411. key_manager.RMA_CHIPLET_TOKEN_DIGEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `token_digest` | R | 0x0 | Hash digest of life cycle secret that allows entering RMA_CHIPLET state/ Note this is a 256 bit field |

### CLASS_KEY

*Table 412. key_manager.CLASS_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key` | R | 0x0 | Class key that could be used for FW encryption/decryption |

### CHIPLET_PUBK_REVOKE

*Table 413. key_manager.CHIPLET_PUBK_REVOKE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `select` | R | 0x0 | Global public-key revocation vector covering every boot-verification key of every owner (classical and PQC). SEP ROM embeds a set of public-key hash digests and accesses additional fused key hashes; each key entry maps to one bit here. Once set, SEP ROM rejects the corresponding key. Bit map: [7:0] ROM chiplet-creator classical keys (bits 0,1 are development keys); [15:8] ROM chiplet-creator PQC keys; [16]/[17] CHIPLET_PUBK_HASH0/1; [18]/[19] CHIPLET_PUBK_PQC_HASH0/1; [20]/[21] SIP_PUBK_HASH0 / SIP_PUBK_PQC_HASH0; [22]/[23] SYS_PUBK_HASH / SYS_PUBK_PQC_HASH; [24]/[25] SIP_PUBK_HASH1 / SIP_PUBK_PQC_HASH1; [31:26] reserved. Before the chiplet owner transfers a part to the customer, the development-key bits shall be set (revoked). |

### BL1_VERSION

*Table 414. key_manager.BL1_VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `version` | R | 0x0 | Minimum BL1 version that can be loaded by SEPROM |

### BL2_VERSION

*Table 415. key_manager.BL2_VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `version` | R | 0x0 | Minimum BL2 version that can be loaded by BL1. If more boot stages exist, rollback protection can be implemented by BL2. |

### CHIPLET_UID

*Table 416. key_manager.CHIPLET_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | Silicon creator's secret device unique ID (also referred to as UDS - Unique Device Secret). The chiplet creator is responsible for CHIPLET_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### SIP_PUBK_HASH0

*Table 417. key_manager.SIP_PUBK_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of SiP owner's root public key for secure boot. The chiplet creator's actual root public key is embedded in the SEPROM; this digest is used to validate that public key. |

### SIP_UID

*Table 418. key_manager.SIP_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | SiP owner's secret device unique ID. The SiP owner is responsible for SIP_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### SYS_PUBK_HASH

*Table 419. key_manager.SYS_PUBK_HASH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Hash digest of system owner's root public key for secure boot. |

### SYS_UID

*Table 420. key_manager.SYS_UID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `uid` | R | 0x0 | System owner's secret device unique ID. The system owner is responsible for SYS_UID provisioning and public identity enrollment. When read-locked, both writability and software visibility are disabled. |

### STATUS_RPT

*Table 421. key_manager.STATUS_RPT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `reserved` | R | 0x0 | reserved |
| 1:0 | `rpt` | R | 0x0 | BOOTROM report status based on the bit. 2'b00→ bootrom report status based on GPIO; 2'b01→bootrom report status;otherwise, bootrom does not report status |

### ROM_CTL

*Table 422. key_manager.ROM_CTL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `reserved` | R | 0x0 | reserved |
| 5:1 | `rom_swap_ctrl` | R | 0x0 | controls how the rom slices are swapped |
| 0 | `rom_endianness_ctrl` | R | 0x0 | controls the endianness of the rom data out. 0→little endian; 1→big endian |

### SYSCLK_FREQ_MHZ

*Table 423. key_manager.SYSCLK_FREQ_MHZ fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `rsvd` | R | 0x0 | Reserved |
| 10:0 | `sysclk_freq_mhz` | R | 0x0 | Configured sysclk PLL frequency in MHz; ignored if 0. Consumed by boot ROM clock math. |

### CHIPLET_PUBK_HASH0

*Table 424. key_manager.CHIPLET_PUBK_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner authentication key hash digest, fused. These chiplet owner fused key hashes can be used in addition to the hashes in the ROM itself to authenticate manifests. |

### CHIPLET_PUBK_HASH1

*Table 425. key_manager.CHIPLET_PUBK_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner authentication key hash digest, fused. These chiplet owner fused key hashes can be used in addition to the hashes in the ROM itself to authenticate manifests. |

### REQUIRED_SIGNERS

*Table 426. key_manager.REQUIRED_SIGNERS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `reserved` | R | 0x0 | Reserved |
| 1:0 | `required_signers` | RW1S | 0x0 | Owners whose signature is required to authenticate any image loadable by SEP ROM. Per owner, 1 = required, 0 = not-required; the default all-zero value requires no additional owner. [0] SiP owner, [1] system owner. The chiplet creator is the root of trust and always authenticates BL1, so it has no bit here. |

### REQUIRED_ALGS

*Table 427. key_manager.REQUIRED_ALGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `reserved` | R | 0x0 | Reserved |
| 11:8 | `sys_algs` | RW1S | 0x0 | Per-owner required signature algorithms for system owner. bit 0 = classical, bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |
| 7:4 | `sip_algs` | RW1S | 0x0 | Per-owner required signature algorithms for SiP owner. bit 0 = classical, bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |
| 3:0 | `chiplet_algs` | RW1S | 0x0 | Per-owner required signature algorithms for chiplet. Each nibble: bit 0 = classical (ECDSA P-384 or RSA), bit 1 = post-quantum (ML-DSA), bits 3:2 reserved. |

### CHIPLET_PUBK_PQC_HASH0

*Table 428. key_manager.CHIPLET_PUBK_PQC_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner ML-DSA PQC authentication key hash digest, fused. PQC counterpart of the corresponding CHIPLET_PUBK_HASH. |

### CHIPLET_PUBK_PQC_HASH1

*Table 429. key_manager.CHIPLET_PUBK_PQC_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `key_hash` | R | 0x0 | Chiplet owner ML-DSA PQC authentication key hash digest, fused. PQC counterpart of the corresponding CHIPLET_PUBK_HASH. |

### SIP_PUBK_PQC_HASH0

*Table 430. key_manager.SIP_PUBK_PQC_HASH0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | key_hash | R | 0x0 | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SIP_PUBK_HASH. |

### SYS_PUBK_PQC_HASH

*Table 431. key_manager.SYS_PUBK_PQC_HASH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | key_hash | R | 0x0 | Hash digest of the system owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SYS_PUBK_HASH. |

### SIP_PUBK_HASH1

*Table 432. key_manager.SIP_PUBK_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | key_hash | R | 0x0 | Hash digest of SiP owner's root public key for secure boot. The chiplet creator's actual root public key is embedded in the SEPROM; this digest is used to validate that public key. |

### SIP_PUBK_PQC_HASH1

*Table 433. key_manager.SIP_PUBK_PQC_HASH1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | key_hash | R | 0x0 | Hash digest of the SiP owner's ML-DSA PQC root public key for secure boot. PQC counterpart of SIP_PUBK_HASH. |

### SEP_CHIPLET_ID

*Table 434. key_manager.SEP_CHIPLET_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | id | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### SEP_SIP_ID

*Table 435. key_manager.SEP_SIP_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | id | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### SEP_SYS_ID

*Table 436. key_manager.SEP_SYS_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | id | R | 0x0 | Per-owner public identity fuse. Personalization input for the Key Manager identity root. Being public, no read protection is required. Individually lockable via LOCKS[58:63] (slots 29-31). |

### spare0

*Table 437. key_manager.spare0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare1

*Table 438. key_manager.spare1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare2

*Table 439. key_manager.spare2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare3

*Table 440. key_manager.spare3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare4

*Table 441. key_manager.spare4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare5

*Table 442. key_manager.spare5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare6

*Table 443. key_manager.spare6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare7

*Table 444. key_manager.spare7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### spare8

*Table 445. key_manager.spare8 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | R | 0x0 | Spare region for future use. Individually lockable via LOCKS_SPARE (slots 32-40). |

### EFUSE_INTERFACE_CTRL_STATUS

*Table 446. key_manager.EFUSE_INTERFACE_CTRL_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10 | `efuse_read_addr_error_clear` | W 1P | 0x0 | Clears the read out of bounds error bit (efuse_read_addr_error) in efuse_ctrl_status |
| 9 | `efuse_program_addr_error_clear` | W 1P | 0x0 | Clears the program out of bounds error bit (efuse_program_addr_error) in efuse_ctrl_status |
| 8 | `efuse_req_error_clear` | W 1P | 0x0 | Clears the blocked request error bit in efuse_ctrl_status |
| 7 | `Reserved` | - | - | Reserved |
| 6 | `efuse_read_addr_error` | R | 0x0 | Indicates that an invalid eFuse read address was used (address exceeds the implemented eFuse size). The error must be cleared by writing 1 to efuse_read_addr_error_clear. |
| 5 | `efuse_program_addr_error` | R | 0x0 | Indicates that an invalid eFuse program address was used (address exceeds the implemented eFuse size). The error must be cleared by writing 1 to efuse_program_addr_error_clear. |
| 4 | `efuse_req_error` | R | 0x0 | Indicates that the eFuse was blocked because of either read or a program lock. The error must be cleared by writing 1 to efuse_req_error_clear to this register |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `efuse_sense_done` | R | - | Indicates if the eFuse state machine has completed |

### EFUSE_PROGRAM_CTRL

*Table 447. key_manager.EFUSE_PROGRAM_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27 | `program_enable` | RW | 0x0 | When set, enables the program request |
| 26 | `program_status` | R | 0x0 | 0: No Error. 1: Error |
| 25 | `program_done` | R | 0x0 | When set, the program has completed, the status of which is in program_status |
| 24 | `program_busy` | R | 0x0 | When set, the program is in progress |
| 23:19 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 18 | `efuse_program_read_back` | RW | 0x0 | When set, trigger a read back after a program |
| 17 | `efuse_program_go` | RW 1P | 0x0 | When set, triggers a program to the eFuse. This will be ignored if the lock bit is set |
| 16 | `efuse_data` | RW | 0x0 | Data to program to eFuse. If this bit is 0, the program request will be ignored |
| 15:0 | `efuse_addr` | RW | 0x0 | Bit address to program |

### EFUSE_READ_CTRL

*Table 448. key_manager.EFUSE_READ_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `read_enable` | RW | 0x0 | When set, enables the read request |
| 27 | `Reserved` | - | - | Reserved |
| 26 | `read_status` | R | 0x0 | 0: No Error. 1: Error. Set when read_go is asserted while read is not enabled, the address is out of bounds, or the read request times out. |
| 25 | `read_done` | R | 0x0 | When set, the read has completed, the status of which is in read_status |
| 24 | `read_busy` | R | 0x0 | When set, the read is in progress |
| 23:17 | `Reserved` | - | - | Reserved |
| 16 | `efuse_read_go` | RW 1P | 0x0 | When set, triggers a read to efuse. |
| 15:0 | `efuse_addr` | RW | 0x0 | Bit address to read |

### EFUSE_PROGRAM_INTERFACE_READ_DATA

*Table 449. key_manager.EFUSE_PROGRAM_INTERFACE_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `dout` | R | 0x0 | Read Data from eFuse through program interface. This field is only valid when the efuse programming interface is utilized to burn the efuse IP and only when the state machine is during ST_CAPTUER_DATA state. Hardware debug only |

### EFUSE_READ_INTERFACE_READ_DATA

*Table 450. key_manager.EFUSE_READ_INTERFACE_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `dout` | R | 0x0 | Read Data from eFuse through READ interface. |

### EFUSE_READ_REQ_TIMEOUT

*Table 451. key_manager.EFUSE_READ_REQ_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `read_req_timout_enable` | RW | 0x0 | Enable timeout on read interface |
| 27:0 | `read_req_timeout_cycles` | RW | 0x800000 | Number of cycles the read interface will wait for a response from the shim. At 100MHz this can count ~= 2.5 seconds. |

### EFUSE_PROGRAM_REQ_TIMEOUT

*Table 452. key_manager.EFUSE_PROGRAM_REQ_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `program_req_timeout_enable` | RW | 0x0 | Enable timeout on program interface. |
| 27:0 | `program_req_timeout_cycles` | RW | 0x800000 | Number of cycles the program interface will wait for a response from the shim. At 100MHz this can count ~= 2.5 seconds. |

### RMA_SIP_TOKEN_I[8]

*Table 453. key_manager.RMA_SIP_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of RMA_*_TOKEN, hashed, and compared with fused value to allow LC to change states. |

### RMA_CHIPLET_TOKEN_I[8]

*Table 454. key_manager.RMA_CHIPLET_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of RMA_*_TOKEN, hashed, and compared with fused value to allow LC to change states. |

### SEC_DISABLE_TOKEN_I[8]

*Table 455. key_manager.SEC_DISABLE_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of SEC_DISABLE_TOKEN, hashed, and compared with the post silicon injected values to allow security_disable. |

### TOKEN_EOP

*Table 456. key_manager.TOKEN_EOP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:17 | `Reserved` | - | - | Reserved |
| 16 | `secure_disable_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `rma_chiplet_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |
| 7:1 | `Reserved` | - | - | Reserved |
| 0 | `rma_sip_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |

### RMA_SIP_TOKEN_MATCH

*Table 457. key_manager.RMA_SIP_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

### RMA_CHIPLET_TOKEN_MATCH

*Table 458. key_manager.RMA_CHIPLET_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

### SEC_DISABLE_TOKEN_MATCH

*Table 459. key_manager.SEC_DISABLE_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

### TOKEN_MATCH_FAULT

*Table 460. key_manager.TOKEN_MATCH_FAULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:17 | `Reserved` | - | - | Reserved |
| 16 | `secure_disable_token_fault` | R | 0x0 | A redundancy fault was detected in the SEC_DISABLE_TOKEN comparators. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `rma_chiplet_token_fault` | R | 0x0 | A redundancy fault was detected in the RMA_CHIPLET_TOKEN comparators. |
| 7:1 | `Reserved` | - | - | Reserved |
| 0 | `rma_sip_token_fault` | R | 0x0 | A redundancy fault was detected in the RMA_SIP_TOKEN comparators. |

### KEY_ENTRY[64].WORD[16]

*Table 461. key_manager.KEY_ENTRY[64].WORD[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | - | 32-bit key data word. No reset for security; power-up value undefined. Actual storage is in km_kpv_regfile; CSR field is a protocol placeholder. |

### CTRL[64]

*Table 462. key_manager.CTRL[64] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `rsvd_31_4` | R | 0x0 | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3 | `seal` | RW1S | 0x0 | Seals this key entry until warm reset. Read-any, write-1-only. Setting this bit also sets lock_write in hardware, so a single write seals the slot: its data can be read but not overwritten. Erase remains available, and erasing a sealed slot retires it rather than freeing it (see ERASE), so a sealed slot's material can be destroyed but its slot never reused. |
| 2 | `erase` | RW1S | 0x0 | Write-1 to start hardware erase of this slot: an LFSR fills all 16 key words (through the KPV scrambler). Hardware self-clears this bit when the erase completes. Never blocked: neither lock_write, lock_use nor seal prevents an erase. The slot's seal state when the erase completes decides the outcome. Unsealed, the slot CTRL register is cleared and the slot is reusable. Sealed, lock_write stays set and lock_use is set, retiring the slot: its data is destroyed and it can be neither read, rewritten nor reused until warm reset. A seal taken while an erase is already in flight therefore still retires the slot. |
| 1 | `lock_use` | RW1S | 0x0 | Prevents KM read of this key entry data until reset. Read-any, write-1-only. Hardware sets this bit when an erase of a sealed slot completes, retiring the slot: its destroyed contents cannot be read back. |
| 0 | `lock_write` | RW1S | 0x0 | Prevents KM write to this key entry data until reset; this register stays writable so that a locked slot can still be read-locked and erased. Read-any, write-1-only. Hardware holds this bit set for as long as SEAL is set, so sealing a slot write-locks it without a separate write here. |

### KPV_SCRAMBLER_KEY

*Table 463. key_manager.KPV_SCRAMBLER_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key` | RW | - | Scrambler key. No reset for security. HW clears to 0 on wipe via hwclr. |

### KPV_SCRAMBLER_CTRL

*Table 464. key_manager.KPV_SCRAMBLER_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `rsvd` | R | 0x0 | Reserved |
| 1 | `lock` | RW1S | 0x0 | Write-one-only. When 1, key and ENABLE cannot be modified until KM reset. |
| 0 | `enable` | RW | 0x0 | 1 = scramble key entry data on write, descramble on read; 0 = passthrough. |

### VERSION

*Table 465. key_manager.VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `rsvd` | R | 0x0 | Reserved; must read as zero |
| 23:16 | `major` | R | 0x1 | Major version number |
| 15:8 | `minor` | R | 0x0 | Minor version number |
| 7:0 | `patch` | R | 0x0 | Patch version number |

### CTRL

*Table 466. key_manager.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `rsvd` | R | 0x0 | Reserved |

### SOFT_RST_CODE

*Table 467. key_manager.SOFT_RST_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `code` | RW | 0x0 | 32-bit reset code. Write 0x53525354 to trigger soft reset. |

### kmcsr.IRQ_STATUS

*Table 468. key_manager.kmcsr.IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | R | 0x0 | Reserved |
| 11 | `rom_access_violation` | RW1C | 0x0 | KM ROM instruction fetch or data read attempted after the ROM lockout engaged. The lockout engages on the first committed instruction fetch from write-locked SRAM once SRAM_EXEC_MODE.enable is set, and clears only on warm or cold reset. Sticky, write 1 to clear. |
| 10 | `exec_violation` | RW1C | 0x0 | Instruction fetch from a non-whitelisted (non-executable) memory region detected, excluding the KM ROM. Sticky, write 1 to clear. |
| 9 | `otp_sigint` | RW1C | 0x0 | OTP dual-rail encoding integrity violation: value and complement are not complementary on at least one field. Sticky, write 1 to clear. |
| 8 | `otp_change` | RW1C | 0x0 | An OTP readout field value changed (decoded value differs from prior cycle). Sticky, write 1 to clear. Check OTP_CHANGE_STATUS for the specific field(s). |
| 7 | `wipe_state` | RW1C | 0x0 | Wipe state event (rising edge of wipe_state input). Sticky, write 1 to clear. |
| 6 | `drbg_err` | RW1C | 0x0 | DRBG Sampler error (timeout or AXI-Stream error). Sticky, write 1 to clear. |
| 5 | `axi_decerr` | RW1C | 0x0 | AXI DECERR (decode error) response detected on CPU bus transaction. Sticky, write 1 to clear. |
| 4 | `axi_slverr` | RW1C | 0x0 | AXI SLVERR (slave error) response detected on CPU bus transaction. Sticky, write 1 to clear. |
| 3 | `sram_write_lock_err` | RW1C | 0x0 | SRAM write attempt to a write-locked region detected. Sticky, write 1 to clear. |
| 2 | `rom_write_err` | RW1C | 0x0 | ROM write attempt detected (ROM is read-only). Sticky, write 1 to clear. |
| 1 | `sram_parity_err` | RW1C | 0x0 | SRAM parity error detected. Sticky, write 1 to clear. |
| 0 | `rom_parity_err` | RW1C | 0x0 | ROM parity error detected. Sticky, write 1 to clear. |

### kmcsr.IRQ_ENABLE

*Table 469. key_manager.kmcsr.IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | R | 0x0 | Reserved |
| 11 | `rom_access_violation_en` | RW | 0x0 | Enable ROM lockout violation interrupt |
| 10 | `exec_violation_en` | RW | 0x0 | Enable execute-permission whitelist violation interrupt |
| 9 | `otp_sigint_en` | RW | 0x0 | Enable OTP dual-rail integrity violation interrupt |
| 8 | `otp_change_en` | RW | 0x0 | Enable OTP field change interrupt |
| 7 | `wipe_state_en` | RW | 0x0 | Enable wipe state interrupt |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 6 | `drbg_err_en` | RW | 0x0 | Enable DRBG Sampler error interrupt |
| 5 | `axi_decerr_en` | RW | 0x0 | Enable AXI DECERR error interrupt |
| 4 | `axi_slverr_en` | RW | 0x0 | Enable AXI SLVERR error interrupt |
| 3 | `sram_write_lock_en` | RW | 0x0 | Enable SRAM write-lock violation interrupt |
| 2 | `rom_write_en` | RW | 0x0 | Enable ROM write error interrupt |
| 1 | `sram_parity_en` | RW | 0x0 | Enable SRAM parity error interrupt |
| 0 | `rom_parity_en` | RW | 0x0 | Enable ROM parity error interrupt |

### SCRAMBLER_KEY

*Table 470. key_manager.SCRAMBLER_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key` | RW | - | Scrambler key value. Not readable when SCRAMBLER_CTRL.LOCK=1 (returns 0). No reset for security; power-up value undefined. |

### SCRAMBLER_CTRL

*Table 471. key_manager.SCRAMBLER_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `rsvd` | R | 0x0 | Reserved |
| 1 | `lock` | RW | 0x0 | Lock scrambler key and enable bit. Once set, key and enable cannot be modified until reset. Write-once (0→1 only). Security requirement: prevents disabling scrambling after key provisioning. |
| 0 | `enable` | RW | 0x0 | Enable SRAM scrambling. When 0, data passes through unmodified. Locked when LOCK=1 (security requirement). |

### SRAM_LOCK

*Table 472. key_manager.SRAM_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `lock_bits` | RW1S | 0x0 | One bit per 1 KB SRAM region. Region 0 = 0x8000-0x83FF, region 31 = 0xFC00-0xFFFF. Write 1 to lock; 0 has no effect. Warm reset domain: cleared by warm or cold reset. |

### IRQ_SET

*Table 473. key_manager.IRQ_SET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | W | 0x0 | Reserved (write-only; reads trigger SLVERR via err_if_bad_rw) |
| 11 | `rom_access_violation_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_ACCESS_VIOLATION (triggers ROM lockout violation interrupt for testing) |
| 10 | `exec_violation_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.EXEC_VIOLATION (triggers exec-permission violation interrupt for testing) |
| 9 | `otp_sigint_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.OTP_SIGINT (triggers OTP sigint interrupt for testing) |
| 8 | `otp_change_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.OTP_CHANGE (triggers OTP change interrupt for testing) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7 | `wipe_state_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.WIPE_STATE (triggers wipe state interrupt for testing) |
| 6 | `drbg_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.DRBG_ERR (triggers DRBG Sampler interrupt for testing) |
| 5 | `axi_decerr_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.AXI_DECERR (triggers AXI DECERR interrupt for testing) |
| 4 | `axi_slverr_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.AXI_SLVERR (triggers AXI SLVERR interrupt for testing) |
| 3 | `sram_write_lock_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.SRAM_WRITE_LOCK_ERR (triggers SRAM write-lock interrupt for testing) |
| 2 | `rom_write_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_WRITE_ERR (triggers ROM write interrupt for testing) |
| 1 | `sram_parity_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.SRAM_PARITY_ERR (triggers SRAM parity interrupt for testing) |
| 0 | `rom_parity_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_PARITY_ERR (triggers ROM parity interrupt for testing) |

### SRAM_WRITE_LOCK_VIOLATION

*Table 474. key_manager.SRAM_WRITE_LOCK_VIOLATION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `violation_bits` | RW1C | 0x0 | Bit[i]=1 if region i had attempted write while locked. Sticky, write 1 to clear each bit. |

### RECOVERABLE_ERR

*Table 475. key_manager.RECOVERABLE_ERR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `recoverable_err` | RW | 0x0 | 1 = recoverable fault occurred and was handled by ISR; 0 = clear. Write 1 to set, write 0 to clear. |

### BOOT_STATUS

*Table 476. key_manager.BOOT_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `cold_boot_done` | RW1S | 0x0 | Write-1-only: set by ROM firmware at end of cold boot. Reads 0 after cold reset; reads 1 after firmware sets it. Sticky until next cold reset (preserved across warm resets). |

### OTP_LIFE_CYCLE

*Table 477. key_manager.OTP_LIFE_CYCLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `rsvd` | R | 0x0 | Reserved |
| 7:0 | `value` | R | - | 8-bit life cycle state from OTP. No reset; power-up value undefined. |

### OTP_DEMOTION_STATE

*Table 478. key_manager.OTP_DEMOTION_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `rsvd` | R | 0x0 | Reserved |
| 3:2 | `demote_2_value` | R | - | 2-bit demotion 2 state from OTP. No reset; power-up value undefined. |
| 1:0 | `demote_1_value` | R | - | 2-bit demotion 1 state from OTP. No reset; power-up value undefined. |

### SRAM_EXEC_MODE

*Table 479. key_manager.SRAM_EXEC_MODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `enable` | RW1S | 0x0 | Execute-permission mode: 0=ROM-only whitelist; 1=write-locked SRAM whitelist, arming the ROM lockout. Write 1 to set; write 0 has no effect. Warm reset domain: cleared by warm or cold reset. |

### IRQ_ENTRY_ADDR

*Table 480. key_manager.IRQ_ENTRY_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `addr` | RW | 0x10 | IRQ handler entry PC. Reset matches ROM vector at 0x10. When locked, SW writes do not update storage. |

### IRQ_ENTRY_LOCK

*Table 481. key_manager.IRQ_ENTRY_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved; reads as zero. |
| 0 | `lock` | RW1S | 0x0 | Write 1 to set; write 0 has no effect; sticky until reset. |

### VUART_TX

*Table 482. key_manager.VUART_TX fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `data_valid` | RW | 0x0 | Data valid strobe. Set by firmware on write, cleared by hardware after one cycle. |
| 30:8 | `rsvd0` | R | 0x0 | Reserved |
| 7:0 | `tx_byte` | RW | 0x0 | Byte to transmit (write to send character to testbench) |

### VUART_RX

*Table 483. key_manager.VUART_RX fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `data_valid` | R | 0x0 | RX data valid. Set by testbench, cleared by firmware read. |
| 30:8 | `rsvd0` | R | 0x0 | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | `rx_byte` | R | 0x0 | Received byte from testbench |

### VUART_STATUS

*Table 484. key_manager.VUART_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `rsvd` | R | 0x0 | Reserved |
| 2 | `print_enable` | R | 0x0 | Enable VUART printing. Set by testbench to enable printf output. Disabled by default to save simulation time. |
| 1 | `rx_valid` | R | 0x0 | RX has valid data available |
| 0 | `tx_ready` | R | 0x1 | TX ready to accept data (always 1 in simulation) |

### TB_RESULT

*Table 485. key_manager.TB_RESULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `result` | RW | 0x0 | Test result: 0=fail, 1=pass |

### TB_SIGNATURE

*Table 486. key_manager.TB_SIGNATURE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `signature` | RW | 0x0 | Completion signature value |

### TB_ERRCODE

*Table 487. key_manager.TB_ERRCODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `errcode` | RW | 0x0 | Error code value |

### TB_SUBTEST

*Table 488. key_manager.TB_SUBTEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `subtest` | RW | 0x0 | Current subtest number |

### TB_CMD

*Table 489. key_manager.TB_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `cmd` | RW | 0x0 | Command code (0=NOP, 1=ROM_PARITY_EN, 2=ROM_PARITY_DIS, 3=SRAM_PARITY_EN, 4=SRAM_PARITY_DIS) |

### TB_CMD_ARG

*Table 490. key_manager.TB_CMD_ARG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `arg` | RW | 0x0 | Command argument value |

### TB_CMD_STATUS

*Table 491. key_manager.TB_CMD_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `status` | RW | 0x0 | Command status: 0=IDLE, 1=ACK, 0xFFFFFFFF=ERR |

### TB_CMD_RESULT

*Table 492. key_manager.TB_CMD_RESULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `result` | R | 0x0 | Command result value |

### DEBUG

*Table 493. key_manager.DEBUG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `magic` | R | 0xCAFEBEEF | Magic constant: 0xCAFEBEEF |

### OTP_CHIPLET_UID_VAL_0

*Table 494. key_manager.OTP_CHIPLET_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_1

*Table 495. key_manager.OTP_CHIPLET_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_2

*Table 496. key_manager.OTP_CHIPLET_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_3

*Table 497. key_manager.OTP_CHIPLET_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_4

*Table 498. key_manager.OTP_CHIPLET_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_5

*Table 499. key_manager.OTP_CHIPLET_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_6

*Table 500. key_manager.OTP_CHIPLET_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_7

*Table 501. key_manager.OTP_CHIPLET_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_CPL_0

*Table 502. key_manager.OTP_CHIPLET_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_CPL_1

*Table 503. key_manager.OTP_CHIPLET_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_CPL_2

*Table 504. key_manager.OTP_CHIPLET_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_3

*Table 505. key_manager.OTP_CHIPLET_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_4

*Table 506. key_manager.OTP_CHIPLET_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_5

*Table 507. key_manager.OTP_CHIPLET_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_6

*Table 508. key_manager.OTP_CHIPLET_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_7

*Table 509. key_manager.OTP_CHIPLET_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_0

*Table 510. key_manager.OTP_SIP_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_1

*Table 511. key_manager.OTP_SIP_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_2

*Table 512. key_manager.OTP_SIP_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_3

*Table 513. key_manager.OTP_SIP_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_4

*Table 514. key_manager.OTP_SIP_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_5

*Table 515. key_manager.OTP_SIP_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_6

*Table 516. key_manager.OTP_SIP_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_7

*Table 517. key_manager.OTP_SIP_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_0

*Table 518. key_manager.OTP_SIP_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_1

*Table 519. key_manager.OTP_SIP_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_2

*Table 520. key_manager.OTP_SIP_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_3

*Table 521. key_manager.OTP_SIP_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_4

*Table 522. key_manager.OTP_SIP_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_5

*Table 523. key_manager.OTP_SIP_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_6

*Table 524. key_manager.OTP_SIP_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_7

*Table 525. key_manager.OTP_SIP_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_0

*Table 526. key_manager.OTP_SYS_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_1

*Table 527. key_manager.OTP_SYS_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_2

*Table 528. key_manager.OTP_SYS_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_3

*Table 529. key_manager.OTP_SYS_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_4

*Table 530. key_manager.OTP_SYS_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_5

*Table 531. key_manager.OTP_SYS_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_6

*Table 532. key_manager.OTP_SYS_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_7

*Table 533. key_manager.OTP_SYS_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_0

*Table 534. key_manager.OTP_SYS_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_1

*Table 535. key_manager.OTP_SYS_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_2

*Table 536. key_manager.OTP_SYS_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_3

*Table 537. key_manager.OTP_SYS_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_4

*Table 538. key_manager.OTP_SYS_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_5

*Table 539. key_manager.OTP_SYS_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_6

*Table 540. key_manager.OTP_SYS_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_7

*Table 541. key_manager.OTP_SYS_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_0

*Table 542. key_manager.OTP_CLASS_KEY_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_1

*Table 543. key_manager.OTP_CLASS_KEY_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_2

*Table 544. key_manager.OTP_CLASS_KEY_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_3

*Table 545. key_manager.OTP_CLASS_KEY_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_4

*Table 546. key_manager.OTP_CLASS_KEY_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_5

*Table 547. key_manager.OTP_CLASS_KEY_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_6

*Table 548. key_manager.OTP_CLASS_KEY_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_7

*Table 549. key_manager.OTP_CLASS_KEY_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_0

*Table 550. key_manager.OTP_CLASS_KEY_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_1

*Table 551. key_manager.OTP_CLASS_KEY_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_2

*Table 552. key_manager.OTP_CLASS_KEY_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_3

*Table 553. key_manager.OTP_CLASS_KEY_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_4

*Table 554. key_manager.OTP_CLASS_KEY_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_5

*Table 555. key_manager.OTP_CLASS_KEY_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_6

*Table 556. key_manager.OTP_CLASS_KEY_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_7

*Table 557. key_manager.OTP_CLASS_KEY_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_READ_LOCK

*Table 558. key_manager.OTP_READ_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1S | 0x0 | Lock OTP_SEP_SYS_ID_VAL/CPL reads (returns zero when locked) |
| 7 | `sep_sip_id` | RW1S | 0x0 | Lock OTP_SEP_SIP_ID_VAL/CPL reads (returns zero when locked) |
| 6 | `sep_chiplet_id` | RW1S | 0x0 | Lock OTP_SEP_CHIPLET_ID_VAL/CPL reads (returns zero when locked) |
| 5 | `class_key` | RW1S | 0x0 | Lock OTP_CLASS_KEY_VAL/CPL reads (returns zero when locked) |
| 4 | `sys_uid` | RW1S | 0x0 | Lock OTP_SYS_UID_VAL/CPL reads (returns zero when locked) |
| 3 | `sip_uid` | RW1S | 0x0 | Lock OTP_SIP_UID_VAL/CPL reads (returns zero when locked) |
| 2 | `chiplet_uid` | RW1S | 0x0 | Lock OTP_CHIPLET_UID_VAL/CPL reads (returns zero when locked) |
| 1 | `demotion` | RW1S | 0x0 | Lock OTP_DEMOTION_STATE reads (returns zero when locked) |
| 0 | `life_cycle` | RW1S | 0x0 | Lock OTP_LIFE_CYCLE reads (returns zero when locked) |

### OTP_CHANGE_STATUS

*Table 559. key_manager.OTP_CHANGE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1C | 0x0 | OTP_SEP_SYS_ID value changed since last cycle. Sticky, write 1 to clear. |
| 7 | `sep_sip_id` | RW1C | 0x0 | OTP_SEP_SIP_ID value changed since last cycle. Sticky, write 1 to clear. |
| 6 | `sep_chiplet_id` | RW1C | 0x0 | OTP_SEP_CHIPLET_ID value changed since last cycle. Sticky, write 1 to clear. |
| 5 | `class_key` | RW1C | 0x0 | OTP_CLASS_KEY value changed since last cycle. Sticky, write 1 to clear. |
| 4 | `sys_uid` | RW1C | 0x0 | OTP_SYS_UID value changed since last cycle. Sticky, write 1 to clear. |
| 3 | `sip_uid` | RW1C | 0x0 | OTP_SIP_UID value changed since last cycle. Sticky, write 1 to clear. |
| 2 | `chiplet_uid` | RW1C | 0x0 | OTP_CHIPLET_UID value changed since last cycle. Sticky, write 1 to clear. |
| 1 | `demotion` | RW1C | 0x0 | OTP_DEMOTION_STATE value changed since last cycle. Sticky, write 1 to clear. |
| 0 | `life_cycle` | RW1C | 0x0 | OTP_LIFE_CYCLE value changed since last cycle. Sticky, write 1 to clear. |

### OTP_READ_LOCK_COLD

*Table 560. key_manager.OTP_READ_LOCK_COLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_SYS_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_sys_id \| OTP_READ_LOCK_COLD.sep_sys_id. |
| 7 | `sep_sip_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_SIP_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_sip_id \| OTP_READ_LOCK_COLD.sep_sip_id. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 6 | `sep_chiplet_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_CHIPLET_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_chiplet_id \| OTP_READ_LOCK_COLD.sep_chiplet_id. |
| 5 | `class_key` | RW1S | 0x0 | Cold-reset read-lock for OTP_CLASS_KEY_VAL/CPL. Effective lock = OTP_READ_LOCK.class_key \| OTP_READ_LOCK_COLD.class_key. |
| 4 | `sys_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_SYS_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.sys_uid \| OTP_READ_LOCK_COLD.sys_uid. |
| 3 | `sip_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_SIP_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.sip_uid \| OTP_READ_LOCK_COLD.sip_uid. |
| 2 | `chiplet_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_CHIPLET_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.chiplet_uid \| OTP_READ_LOCK_COLD.chiplet_uid. |
| 1 | `demotion` | RW1S | 0x0 | Cold-reset read-lock for OTP_DEMOTION_STATE. Effective lock = OTP_READ_LOCK.demotion \| OTP_READ_LOCK_COLD.demotion. |
| 0 | `life_cycle` | RW1S | 0x0 | Cold-reset read-lock for OTP_LIFE_CYCLE. Effective lock = OTP_READ_LOCK.life_cycle \| OTP_READ_LOCK_COLD.life_cycle. |

### OTP_SEP_CHIPLET_ID_VAL_0

*Table 561. key_manager.OTP_SEP_CHIPLET_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_1

*Table 562. key_manager.OTP_SEP_CHIPLET_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_2

*Table 563. key_manager.OTP_SEP_CHIPLET_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_3

*Table 564. key_manager.OTP_SEP_CHIPLET_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_4

*Table 565. key_manager.OTP_SEP_CHIPLET_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_VAL_5

*Table 566. key_manager.OTP_SEP_CHIPLET_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_VAL_6

*Table 567. key_manager.OTP_SEP_CHIPLET_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_VAL_7

*Table 568. key_manager.OTP_SEP_CHIPLET_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_0

*Table 569. key_manager.OTP_SEP_CHIPLET_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_1

*Table 570. key_manager.OTP_SEP_CHIPLET_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_2

*Table 571. key_manager.OTP_SEP_CHIPLET_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_3

*Table 572. key_manager.OTP_SEP_CHIPLET_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_4

*Table 573. key_manager.OTP_SEP_CHIPLET_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_5

*Table 574. key_manager.OTP_SEP_CHIPLET_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_6

*Table 575. key_manager.OTP_SEP_CHIPLET_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_7

*Table 576. key_manager.OTP_SEP_CHIPLET_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_0

*Table 577. key_manager.OTP_SEP_SIP_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_1

*Table 578. key_manager.OTP_SEP_SIP_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_2

*Table 579. key_manager.OTP_SEP_SIP_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_3

*Table 580. key_manager.OTP_SEP_SIP_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_4

*Table 581. key_manager.OTP_SEP_SIP_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_5

*Table 582. key_manager.OTP_SEP_SIP_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_6

*Table 583. key_manager.OTP_SEP_SIP_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_7

*Table 584. key_manager.OTP_SEP_SIP_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_0

*Table 585. key_manager.OTP_SEP_SIP_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_1

*Table 586. key_manager.OTP_SEP_SIP_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_2

*Table 587. key_manager.OTP_SEP_SIP_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_3

*Table 588. key_manager.OTP_SEP_SIP_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_4

*Table 589. key_manager.OTP_SEP_SIP_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_5

*Table 590. key_manager.OTP_SEP_SIP_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_6

*Table 591. key_manager.OTP_SEP_SIP_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_CPL_7

*Table 592. key_manager.OTP_SEP_SIP_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_0

*Table 593. key_manager.OTP_SEP_SYS_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_1

*Table 594. key_manager.OTP_SEP_SYS_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_2

*Table 595. key_manager.OTP_SEP_SYS_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_3

*Table 596. key_manager.OTP_SEP_SYS_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_4

*Table 597. key_manager.OTP_SEP_SYS_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_5

*Table 598. key_manager.OTP_SEP_SYS_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_6

*Table 599. key_manager.OTP_SEP_SYS_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_7

*Table 600. key_manager.OTP_SEP_SYS_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_0

*Table 601. key_manager.OTP_SEP_SYS_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_1

*Table 602. key_manager.OTP_SEP_SYS_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_2

*Table 603. key_manager.OTP_SEP_SYS_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_CPL_3

*Table 604. key_manager.OTP_SEP_SYS_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_CPL_4

*Table 605. key_manager.OTP_SEP_SYS_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_CPL_5

*Table 606. key_manager.OTP_SEP_SYS_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_CPL_6

*Table 607. key_manager.OTP_SEP_SYS_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_CPL_7

*Table 608. key_manager.OTP_SEP_SYS_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### DATA

*Table 609. key_manager.DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | 32-bit random data from DRBG. No reset for security; power-up value undefined. |

### CFG

*Table 610. key_manager.CFG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `timeout` | RW | 0x100 | Cycles to wait for DRBG on active CPU read; 0 = disable. Default 256. |
| 15:1 | `rsvd` | RW | 0x0 | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `prefetch` | RW | 0x0 | Enable prefetch. When 0, prefetch data register is cleared. |

### STATUS

*Table 611. key_manager.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `count_good` | RW | 0x0 | Counts only reads completed through ST_RESPOND, excluding prefetched words. Saturates at 0xFFFF. Writing any non-zero value clears the field to 0; writing 0 has no effect. |
| 15:8 | `count_bad` | RW | 0x0 | Failed transactions (timeout or error). Saturates at 0xFF. Writing any non-zero value clears the field to 0; writing 0 has no effect. |
| 7:4 | `rsvd` | R | 0x0 | Reserved |
| 3 | `stream_err` | RW1C | 0x0 | Set when there is an error on the DRBG AXI-Stream interface (e.g. TVALID deasserted before TREADY). Write 1 to clear. |
| 2 | `timeout_err` | RW1C | 0x0 | Set when a DRBG read from the KM CPU times out. Write 1 to clear. |
| 1 | `prefetched` | R | 0x0 | Prefetched data available |
| 0 | `drbg_ready` | R | 0x0 | TVALID asserted from DRBG |

### PREFETCH_DATA

*Table 612. key_manager.PREFETCH_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | Prefetched 32-bit random data. No reset for security; power-up value undefined. |

### KEY_SHARE0[12]

*Table 613. key_manager.KEY_SHARE0[12] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### KEY_SHARE1[12]

*Table 614. key_manager.KEY_SHARE1[12] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### otbn_wrapper_key.KEY_CTRL

*Table 615. key_manager.otbn_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### aes_wrapper_key.KEY_SHARE0[8]

*Table 616. key_manager.aes_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### aes_wrapper_key.KEY_SHARE1[8]

*Table 617. key_manager.aes_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### aes_wrapper_key.KEY_CTRL

*Table 618. key_manager.aes_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### kmac_wrapper_key.KEY_SHARE0[8]

*Table 619. key_manager.kmac_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### kmac_wrapper_key.KEY_SHARE1[8]

*Table 620. key_manager.kmac_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### kmac_wrapper_key.KEY_CTRL

*Table 621. key_manager.kmac_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### hmac_wrapper_key.KEY_SHARE0[8]

*Table 622. key_manager.hmac_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### hmac_wrapper_key.KEY_SHARE1[8]

*Table 623. key_manager.hmac_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### hmac_wrapper_key.KEY_CTRL

*Table 624. key_manager.hmac_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8]

*Table 625. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8]

*Table 626. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLDSA_SEED.KEY_CTRL

*Table 627. key_manager.abr_wrapper_key.MLDSA_SEED.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

### abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8]

*Table 628. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8]

*Table 629. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL

*Table 630. key_manager.abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

### abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8]

*Table 631. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8]

*Table 632. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL

*Table 633. key_manager.abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

### abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8]

*Table 634. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8]

*Table 635. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### abr_wrapper_key.MLKEM_MSG.KEY_CTRL

*Table 636. key_manager.abr_wrapper_key.MLKEM_MSG.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

#### KEY[8]

*Table 637. key_manager.KEY[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | 0x0 | 32-bit shared-key word. Hardware writes on key-ready event; hardware clears to 0 on key-consume event (KEY_VALID cleared). |

#### abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL

*Table 638. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW0C | 0x0 | 1 = shared key is valid and ready for firmware consumption. Hardware sets this bit on key-ready event. Write 0 to clear; write 1 has no effect (software cannot set this bit). |

#### abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_STATUS

*Table 639. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW1C | 0x0 | ML-KEM shared key valid. Sticky, write 1 to clear. |

#### abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_ENABLE

*Table 640. key_manager.abr_wrapper_key.MLKEM_SHARED_KEY.IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid_en` | RW | 0x0 | 1 = enable PicoRV32 IRQ when ML-KEM shared key is valid. 0 = mask the interrupt (KEY_CTRL.KEY_VALID can still be polled). |

### Control and Status Registers

#### Key Manager Control and Status Registers

Control and status registers for the Key Manager subsystem

*Table 641. Key Manager Control and Status Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | VERSION | R | Hardware version register (semantic versioning: major.minor.patch) |
| 0x4 | CTRL | R | Control register (reserved for future use) |
| 0x8 | SOFT_RST_CODE | RW | Software reset code register. Write 0x53525354 ('SRST') to trigger soft reset. Any other value has no effect. |
| 0xC | IRQ_STATUS | RW | Interrupt status register. Sticky bits cleared by writing 1. |
| 0x10 | IRQ_ENABLE | RW | Interrupt enable/mask register |
| 0x14 | SCRAMBLER_KEY | RW | 32-bit scrambler key for SRAM address/data scrambling. When locked: writes ignored, reads return 0. No reset; powers up random. |
| 0x18 | SCRAMBLER_CTRL | RW | Scrambler control register |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1C | SRAM_LOCK | RW | SRAM write-lock bits. Bit[i]=1 locks region i (1 KB each, 32 regions). Write-1-only: writing 1 sets the bit, writing 0 has no effect. Cleared by warm or cold reset (warm reset domain); ROM re-applies the lock policy on every boot before handover to mutable firmware. |
| 0x20 | IRQ_SET | W | Interrupt set register. Write 1 to trigger corresponding interrupt (for ISR testing). Write-only, reads return 0. |
| 0x24 | SRAM_WRITE_LOCK_VIOLATION | RW | Which SRAM region(s) had write attempts while locked. Bit[i]=region i. Sticky, write 1 to clear each bit. |
| 0x28 | RECOVERABLE_ERR | RW | Recoverable error status. Firmware writes 1 to set (after recovering in ISR), writes 0 to clear (e.g. when SEP requests clear). Drives recoverable error event output. |
| 0x2C | BOOT_STATUS | RW | Boot status. Firmware writes 1 to COLD_BOOT_DONE after cold-boot init completes; bit remains set until next cold reset. |
| 0x30 | OTP_LIFE_CYCLE | R | Life cycle state. 8-bit (4-bit value differentially encoded). Read-through from OTP port; no reset. |
| 0x34 | OTP_DEMOTION_STATE | R | Demotion state. 2-bit, differentially encoded. Read-through from OTP port; no reset. |
| 0x38 | SRAM_EXEC_MODE | RW | Execute-permission whitelist mode. Write-1-only (woset); cleared by warm or cold reset. 0=ROM executable; 1=write-locked SRAM regions executable, and the ROM lockout is armed. Set by ROM stack-less handover immediately before jumping to mutable firmware. |
| 0xB8 | IRQ_ENTRY_ADDR | RW | Effective IRQ entry address for subsequent interrupts. Writable while unlocked; any 32-bit value accepted. |
| 0xBC | IRQ_ENTRY_LOCK | RW | Write-one-only lock for IRQ_ENTRY_ADDR. Separate readable register; cleared only on KM reset. |
| 0x100 | VUART_TX | RW | Virtual UART transmit register. Write a byte to send to testbench. DATA_VALID is set on write and self-clears. |
| 0x104 | VUART_RX | R | Virtual UART receive register. Testbench writes bytes here for firmware to read. |
| 0x108 | VUART_STATUS | R | Virtual UART status register for flow control |
| 0x110 | TB_RESULT | RW | Test result register. Firmware writes 0=fail, 1=pass. |
| 0x114 | TB_SIGNATURE | RW | Test completion signature. Firmware writes 0x600D600D (pass) or 0xBADBADBA (fail). |
| 0x118 | TB_ERRCODE | RW | Test error code register. Firmware writes optional error code for debugging. |
| 0x11C | TB_SUBTEST | RW | Current subtest number register. Firmware increments for each subtest. |
| 0x120 | TB_CMD | RW | Testbench command register. Firmware writes command, testbench reads and clears. |
| 0x124 | TB_CMD_ARG | RW | Testbench command argument register. Firmware writes optional argument. |
| 0x128 | TB_CMD_STATUS | RW | Testbench command status register. Testbench writes status, firmware reads. |
| 0x12C | TB_CMD_RESULT | R | Testbench command result register. Testbench writes result value. |
| 0x1FC | DEBUG | R | Debug register with known constant value for verification |
| 0x200 | OTP_CHIPLET_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x204 | OTP_CHIPLET_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x208 | OTP_CHIPLET_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x20C | OTP_CHIPLET_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x210 | OTP_CHIPLET_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x214 | OTP_CHIPLET_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x218 | OTP_CHIPLET_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x21C | OTP_CHIPLET_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x220 | OTP_CHIPLET_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x224 | OTP_CHIPLET_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x228 | OTP_CHIPLET_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x22C | OTP_CHIPLET_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x230 | OTP_CHIPLET_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x234 | OTP_CHIPLET_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x238 | OTP_CHIPLET_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x23C | OTP_CHIPLET_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x240 | OTP_SIP_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x244 | OTP_SIP_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x248 | OTP_SIP_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x24C | OTP_SIP_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x250 | OTP_SIP_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x254 | OTP_SIP_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x258 | OTP_SIP_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x25C | OTP_SIP_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x260 | OTP_SIP_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x264 | OTP_SIP_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x268 | OTP_SIP_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x26C | OTP_SIP_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x270 | OTP_SIP_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x274 | OTP_SIP_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x278 | OTP_SIP_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x27C | OTP_SIP_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x280 | OTP_SYS_UID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x284 | OTP_SYS_UID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x288 | OTP_SYS_UID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x28C | OTP_SYS_UID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x290 | OTP_SYS_UID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x294 | OTP_SYS_UID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x298 | OTP_SYS_UID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x29C | OTP_SYS_UID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2A0 | OTP_SYS_UID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2A4 | OTP_SYS_UID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2A8 | OTP_SYS_UID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2AC | OTP_SYS_UID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2B0 | OTP_SYS_UID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2B4 | OTP_SYS_UID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2B8 | OTP_SYS_UID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2BC | OTP_SYS_UID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2C0 | OTP_CLASS_KEY_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2C4 | OTP_CLASS_KEY_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2C8 | OTP_CLASS_KEY_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2CC | OTP_CLASS_KEY_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2D0 | OTP_CLASS_KEY_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2D4 | OTP_CLASS_KEY_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2D8 | OTP_CLASS_KEY_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2DC | OTP_CLASS_KEY_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2E0 | OTP_CLASS_KEY_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2E4 | OTP_CLASS_KEY_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2E8 | OTP_CLASS_KEY_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2EC | OTP_CLASS_KEY_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2F0 | OTP_CLASS_KEY_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2F4 | OTP_CLASS_KEY_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2F8 | OTP_CLASS_KEY_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x2FC | OTP_CLASS_KEY_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x300 | OTP_READ_LOCK | RW | OTP field warm-reset-domain read-lock bits. Write 1 to set; writing 0 has no effect. Locked reads of the corresponding OTP field return zero (effective lock = OTP_READ_LOCK \| OTP_READ_LOCK_COLD). Warm-reset domain: cleared by warm or cold reset. |
| 0x304 | OTP_CHANGE_STATUS | RW | OTP field change status. A bit is set when the corresponding field's decoded value changes. Sticky, write 1 to clear. Valid after IRQ_STATUS.OTP_CHANGE fires. |
| 0x308 | OTP_READ_LOCK_COLD | RW | OTP field cold-reset-domain read-lock bits. Write 1 to set; writing 0 has no effect. Locked reads of the corresponding OTP field return zero (effective lock = OTP_READ_LOCK \| OTP_READ_LOCK_COLD). Cold-reset domain: survives warm reset. |
| 0x400 | OTP_SEP_CHIPLET_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x404 | OTP_SEP_CHIPLET_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x408 | OTP_SEP_CHIPLET_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x40C | OTP_SEP_CHIPLET_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x410 | OTP_SEP_CHIPLET_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x414 | OTP_SEP_CHIPLET_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x418 | OTP_SEP_CHIPLET_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x41C | OTP_SEP_CHIPLET_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x420 | OTP_SEP_CHIPLET_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x424 | OTP_SEP_CHIPLET_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x428 | OTP_SEP_CHIPLET_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x42C | OTP_SEP_CHIPLET_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x430 | OTP_SEP_CHIPLET_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x434 | OTP_SEP_CHIPLET_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x438 | OTP_SEP_CHIPLET_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x43C | OTP_SEP_CHIPLET_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x440 | OTP_SEP_SIP_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x444 | OTP_SEP_SIP_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x448 | OTP_SEP_SIP_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x44C | OTP_SEP_SIP_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x450 | OTP_SEP_SIP_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x454 | OTP_SEP_SIP_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x458 | OTP_SEP_SIP_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x45C | OTP_SEP_SIP_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x460 | OTP_SEP_SIP_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x464 | OTP_SEP_SIP_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x468 | OTP_SEP_SIP_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x46C | OTP_SEP_SIP_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x470 | OTP_SEP_SIP_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x474 | OTP_SEP_SIP_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x478 | OTP_SEP_SIP_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x47C | OTP_SEP_SIP_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x480 | OTP_SEP_SYS_ID_VAL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x484 | OTP_SEP_SYS_ID_VAL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x488 | OTP_SEP_SYS_ID_VAL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x48C | OTP_SEP_SYS_ID_VAL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x490 | OTP_SEP_SYS_ID_VAL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x494 | OTP_SEP_SYS_ID_VAL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x498 | OTP_SEP_SYS_ID_VAL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x49C | OTP_SEP_SYS_ID_VAL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4A0 | OTP_SEP_SYS_ID_CPL_0 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4A4 | OTP_SEP_SYS_ID_CPL_1 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x4A8 | OTP_SEP_SYS_ID_CPL_2 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4AC | OTP_SEP_SYS_ID_CPL_3 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4B0 | OTP_SEP_SYS_ID_CPL_4 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4B4 | OTP_SEP_SYS_ID_CPL_5 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4B8 | OTP_SEP_SYS_ID_CPL_6 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |
| 0x4BC | OTP_SEP_SYS_ID_CPL_7 | R | Dual-rail OTP readout word (value or complement half). Read-through from OTP port; no reset. Returns zero when the field's bit is set in either OTP_READ_LOCK or OTP_READ_LOCK_COLD. |

### VERSION

*Table 642. km_csr.VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `rsvd` | R | 0x0 | Reserved; must read as zero |
| 23:16 | `major` | R | 0x1 | Major version number |
| 15:8 | `minor` | R | 0x0 | Minor version number |
| 7:0 | `patch` | R | 0x0 | Patch version number |

### CTRL

*Table 643. km_csr.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `rsvd` | R | 0x0 | Reserved |

### SOFT_RST_CODE

*Table 644. km_csr.SOFT_RST_CODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `code` | RW | 0x0 | 32-bit reset code. Write 0x53525354 to trigger soft reset. |

### IRQ_STATUS

*Table 645. km_csr.IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | R | 0x0 | Reserved |
| 11 | `rom_access_violation` | RW1C | 0x0 | KM ROM instruction fetch or data read attempted after the ROM lockout engaged. The lockout engages on the first committed instruction fetch from write-locked SRAM once SRAM_EXEC_MODE.enable is set, and clears only on warm or cold reset. Sticky, write 1 to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 10 | `exec_violation` | RW1C | 0x0 | Instruction fetch from a non-whitelisted (non-executable) memory region detected, excluding the KM ROM. Sticky, write 1 to clear. |
| 9 | `otp_sigint` | RW1C | 0x0 | OTP dual-rail encoding integrity violation: value and complement are not complementary on at least one field. Sticky, write 1 to clear. |
| 8 | `otp_change` | RW1C | 0x0 | An OTP readout field value changed (decoded value differs from prior cycle). Sticky, write 1 to clear. Check OTP_CHANGE_STATUS for the specific field(s). |
| 7 | `wipe_state` | RW1C | 0x0 | Wipe state event (rising edge of wipe_state input). Sticky, write 1 to clear. |
| 6 | `drbg_err` | RW1C | 0x0 | DRBG Sampler error (timeout or AXI-Stream error). Sticky, write 1 to clear. |
| 5 | `axi_decerr` | RW1C | 0x0 | AXI DECERR (decode error) response detected on CPU bus transaction. Sticky, write 1 to clear. |
| 4 | `axi_slverr` | RW1C | 0x0 | AXI SLVERR (slave error) response detected on CPU bus transaction. Sticky, write 1 to clear. |
| 3 | `sram_write_lock_err` | RW1C | 0x0 | SRAM write attempt to a write-locked region detected. Sticky, write 1 to clear. |
| 2 | `rom_write_err` | RW1C | 0x0 | ROM write attempt detected (ROM is read-only). Sticky, write 1 to clear. |
| 1 | `sram_parity_err` | RW1C | 0x0 | SRAM parity error detected. Sticky, write 1 to clear. |
| 0 | `rom_parity_err` | RW1C | 0x0 | ROM parity error detected. Sticky, write 1 to clear. |

### IRQ_ENABLE

*Table 646. km_csr.IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | R | 0x0 | Reserved |
| 11 | `rom_access_violation_en` | RW | 0x0 | Enable ROM lockout violation interrupt |
| 10 | `exec_violation_en` | RW | 0x0 | Enable execute-permission whitelist violation interrupt |
| 9 | `otp_sigint_en` | RW | 0x0 | Enable OTP dual-rail integrity violation interrupt |
| 8 | `otp_change_en` | RW | 0x0 | Enable OTP field change interrupt |
| 7 | `wipe_state_en` | RW | 0x0 | Enable wipe state interrupt |
| 6 | `drbg_err_en` | RW | 0x0 | Enable DRBG Sampler error interrupt |
| 5 | `axi_decerr_en` | RW | 0x0 | Enable AXI DECERR error interrupt |
| 4 | `axi_slverr_en` | RW | 0x0 | Enable AXI SLVERR error interrupt |
| 3 | `sram_write_lock_en` | RW | 0x0 | Enable SRAM write-lock violation interrupt |
| 2 | `rom_write_en` | RW | 0x0 | Enable ROM write error interrupt |
| 1 | `sram_parity_en` | RW | 0x0 | Enable SRAM parity error interrupt |
| 0 | `rom_parity_en` | RW | 0x0 | Enable ROM parity error interrupt |

### SCRAMBLER_KEY

*Table 647. km_csr.SCRAMBLER_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key` | RW | - | Scrambler key value. Not readable when SCRAMBLER_CTRL.LOCK=1 (returns 0). No reset for security; power-up value undefined. |

### SCRAMBLER_CTRL

*Table 648. km_csr.SCRAMBLER_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `rsvd` | R | 0x0 | Reserved |
| 1 | `lock` | RW | 0x0 | Lock scrambler key and enable bit. Once set, key and enable cannot be modified until reset. Write-once (0→1 only). Security requirement: prevents disabling scrambling after key provisioning. |
| 0 | `enable` | RW | 0x0 | Enable SRAM scrambling. When 0, data passes through unmodified. Locked when LOCK=1 (security requirement). |

### SRAM_LOCK

*Table 649. km_csr.SRAM_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `lock_bits` | RW1S | 0x0 | One bit per 1 KB SRAM region. Region 0 = 0x8000-0x83FF, region 31 = 0xFC00-0xFFFF. Write 1 to lock; 0 has no effect. Warm reset domain: cleared by warm or cold reset. |

### IRQ_SET

*Table 650. km_csr.IRQ_SET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `rsvd` | W | 0x0 | Reserved (write-only; reads trigger SLVERR via err_if_bad_rw) |
| 11 | `rom_access_violation_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_ACCESS_VIOLATION (triggers ROM lockout violation interrupt for testing) |
| 10 | `exec_violation_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.EXEC_VIOLATION (triggers exec-permission violation interrupt for testing) |
| 9 | `otp_sigint_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.OTP_SIGINT (triggers OTP sigint interrupt for testing) |
| 8 | `otp_change_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.OTP_CHANGE (triggers OTP change interrupt for testing) |
| 7 | `wipe_state_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.WIPE_STATE (triggers wipe state interrupt for testing) |
| 6 | `drbg_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.DRBG_ERR (triggers DRBG Sampler interrupt for testing) |
| 5 | `axi_decerr_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.AXI_DECERR (triggers AXI DECERR interrupt for testing) |
| 4 | `axi_slverr_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.AXI_SLVERR (triggers AXI SLVERR interrupt for testing) |
| 3 | `sram_write_lock_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.SRAM_WRITE_LOCK_ERR (triggers SRAM write-lock interrupt for testing) |
| 2 | `rom_write_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_WRITE_ERR (triggers ROM write interrupt for testing) |
| 1 | `sram_parity_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.SRAM_PARITY_ERR (triggers SRAM parity interrupt for testing) |
| 0 | `rom_parity_err_set` | W 1P | 0x0 | Write 1 to set IRQ_STATUS.ROM_PARITY_ERR (triggers ROM parity interrupt for testing) |

### SRAM_WRITE_LOCK_VIOLATION

*Table 651. km_csr.SRAM_WRITE_LOCK_VIOLATION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `violation_bits` | RW1C | 0x0 | Bit[i]=1 if region i had attempted write while locked. Sticky, write 1 to clear each bit. |

### RECOVERABLE_ERR

*Table 652. km_csr.RECOVERABLE_ERR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `recoverable_err` | RW | 0x0 | 1 = recoverable fault occurred and was handled by ISR; 0 = clear. Write 1 to set, write 0 to clear. |

### BOOT_STATUS

*Table 653. km_csr.BOOT_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `cold_boot_done` | RW1S | 0x0 | Write-1-only: set by ROM firmware at end of cold boot. Reads 0 after cold reset; reads 1 after firmware sets it. Sticky until next cold reset (preserved across warm resets). |

### OTP_LIFE_CYCLE

*Table 654. km_csr.OTP_LIFE_CYCLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `rsvd` | R | 0x0 | Reserved |
| 7:0 | `value` | R | - | 8-bit life cycle state from OTP. No reset; power-up value undefined. |

### OTP_DEMOTION_STATE

*Table 655. km_csr.OTP_DEMOTION_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `rsvd` | R | 0x0 | Reserved |
| 3:2 | `demote_2_value` | R | - | 2-bit demotion 2 state from OTP. No reset; power-up value undefined. |
| 1:0 | `demote_1_value` | R | - | 2-bit demotion 1 state from OTP. No reset; power-up value undefined. |

### SRAM_EXEC_MODE

*Table 656. km_csr.SRAM_EXEC_MODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `enable` | RW1S | 0x0 | Execute-permission mode: 0=ROM-only whitelist; 1=write-locked SRAM whitelist, arming the ROM lockout. Write 1 to set; write 0 has no effect. Warm reset domain: cleared by warm or cold reset. |

### IRQ_ENTRY_ADDR

*Table 657. km_csr.IRQ_ENTRY_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `addr` | RW | 0x10 | IRQ handler entry PC. Reset matches ROM vector at 0x10. When locked, SW writes do not update storage. |

### IRQ_ENTRY_LOCK

*Table 658. km_csr.IRQ_ENTRY_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved; reads as zero. |
| 0 | `lock` | RW1S | 0x0 | Write 1 to set; write 0 has no effect; sticky until reset. |

### VUART_TX

*Table 659. km_csr.VUART_TX fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `data_valid` | RW | 0x0 | Data valid strobe. Set by firmware on write, cleared by hardware after one cycle. |
| 30:8 | `rsvd0` | R | 0x0 | Reserved |
| 7:0 | `tx_byte` | RW | 0x0 | Byte to transmit (write to send character to testbench) |

### VUART_RX

*Table 660. km_csr.VUART_RX fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `data_valid` | R | 0x0 | RX data valid. Set by testbench, cleared by firmware read. |
| 30:8 | `rsvd0` | R | 0x0 | Reserved |
| 7:0 | `rx_byte` | R | 0x0 | Received byte from testbench |

### VUART_STATUS

*Table 661. km_csr.VUART_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `rsvd` | R | 0x0 | Reserved |
| 2 | `print_enable` | R | 0x0 | Enable VUART printing. Set by testbench to enable printf output. Disabled by default to save simulation time. |
| 1 | `rx_valid` | R | 0x0 | RX has valid data available |
| 0 | `tx_ready` | R | 0x1 | TX ready to accept data (always 1 in simulation) |

### TB_RESULT

*Table 662. km_csr.TB_RESULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `result` | RW | 0x0 | Test result: 0=fail, 1=pass |

### TB_SIGNATURE

*Table 663. km_csr.TB_SIGNATURE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `signature` | RW | 0x0 | Completion signature value |

### TB_ERRCODE

*Table 664. km_csr.TB_ERRCODE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `errcode` | RW | 0x0 | Error code value |

### TB_SUBTEST

*Table 665. km_csr.TB_SUBTEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `subtest` | RW | 0x0 | Current subtest number |

### TB_CMD

*Table 666. km_csr.TB_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `cmd` | RW | 0x0 | Command code (0=NOP, 1=ROM_PARITY_EN, 2=ROM_PARITY_DIS, 3=SRAM_PARITY_EN, 4=SRAM_PARITY_DIS) |

### TB_CMD_ARG

*Table 667. km_csr.TB_CMD_ARG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `arg` | RW | 0x0 | Command argument value |

### TB_CMD_STATUS

*Table 668. km_csr.TB_CMD_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `status` | RW | 0x0 | Command status: 0=IDLE, 1=ACK, 0xFFFFFFFF=ERR |

### TB_CMD_RESULT

*Table 669. km_csr.TB_CMD_RESULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `result` | R | 0x0 | Command result value |

### DEBUG

*Table 670. km_csr.DEBUG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `magic` | R | 0xCAFEBEEF | Magic constant: 0xCAFEBEEF |

### OTP_CHIPLET_UID_VAL_0

*Table 671. km_csr.OTP_CHIPLET_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_1

*Table 672. km_csr.OTP_CHIPLET_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_2

*Table 673. km_csr.OTP_CHIPLET_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_3

*Table 674. km_csr.OTP_CHIPLET_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_4

*Table 675. km_csr.OTP_CHIPLET_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_5

*Table 676. km_csr.OTP_CHIPLET_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_VAL_6

*Table 677. km_csr.OTP_CHIPLET_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_VAL_7

*Table 678. km_csr.OTP_CHIPLET_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_0

*Table 679. km_csr.OTP_CHIPLET_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_1

*Table 680. km_csr.OTP_CHIPLET_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_2

*Table 681. km_csr.OTP_CHIPLET_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_3

*Table 682. km_csr.OTP_CHIPLET_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_4

*Table 683. km_csr.OTP_CHIPLET_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_5

*Table 684. km_csr.OTP_CHIPLET_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CHIPLET_UID_CPL_6

*Table 685. km_csr.OTP_CHIPLET_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CHIPLET_UID_CPL_7

*Table 686. km_csr.OTP_CHIPLET_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_0

*Table 687. km_csr.OTP_SIP_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_1

*Table 688. km_csr.OTP_SIP_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_2

*Table 689. km_csr.OTP_SIP_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_3

*Table 690. km_csr.OTP_SIP_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_4

*Table 691. km_csr.OTP_SIP_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_VAL_5

*Table 692. km_csr.OTP_SIP_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_6

*Table 693. km_csr.OTP_SIP_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_VAL_7

*Table 694. km_csr.OTP_SIP_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_0

*Table 695. km_csr.OTP_SIP_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_1

*Table 696. km_csr.OTP_SIP_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_2

*Table 697. km_csr.OTP_SIP_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_3

*Table 698. km_csr.OTP_SIP_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_4

*Table 699. km_csr.OTP_SIP_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SIP_UID_CPL_5

*Table 700. km_csr.OTP_SIP_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_CPL_6

*Table 701. km_csr.OTP_SIP_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SIP_UID_CPL_7

*Table 702. km_csr.OTP_SIP_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SYS_UID_VAL_0

*Table 703. km_csr.OTP_SYS_UID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SYS_UID_VAL_1

*Table 704. km_csr.OTP_SYS_UID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SYS_UID_VAL_2

*Table 705. km_csr.OTP_SYS_UID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SYS_UID_VAL_3

*Table 706. km_csr.OTP_SYS_UID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SYS_UID_VAL_4

*Table 707. km_csr.OTP_SYS_UID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_5

*Table 708. km_csr.OTP_SYS_UID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_6

*Table 709. km_csr.OTP_SYS_UID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_VAL_7

*Table 710. km_csr.OTP_SYS_UID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_0

*Table 711. km_csr.OTP_SYS_UID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_1

*Table 712. km_csr.OTP_SYS_UID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_2

*Table 713. km_csr.OTP_SYS_UID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_3

*Table 714. km_csr.OTP_SYS_UID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_4

*Table 715. km_csr.OTP_SYS_UID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_5

*Table 716. km_csr.OTP_SYS_UID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_6

*Table 717. km_csr.OTP_SYS_UID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SYS_UID_CPL_7

*Table 718. km_csr.OTP_SYS_UID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_0

*Table 719. km_csr.OTP_CLASS_KEY_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_1

*Table 720. km_csr.OTP_CLASS_KEY_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_2

*Table 721. km_csr.OTP_CLASS_KEY_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_3

*Table 722. km_csr.OTP_CLASS_KEY_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_4

*Table 723. km_csr.OTP_CLASS_KEY_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_5

*Table 724. km_csr.OTP_CLASS_KEY_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_6

*Table 725. km_csr.OTP_CLASS_KEY_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_VAL_7

*Table 726. km_csr.OTP_CLASS_KEY_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_0

*Table 727. km_csr.OTP_CLASS_KEY_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_1

*Table 728. km_csr.OTP_CLASS_KEY_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_2

*Table 729. km_csr.OTP_CLASS_KEY_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_CLASS_KEY_CPL_3

*Table 730. km_csr.OTP_CLASS_KEY_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CLASS_KEY_CPL_4

*Table 731. km_csr.OTP_CLASS_KEY_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CLASS_KEY_CPL_5

*Table 732. km_csr.OTP_CLASS_KEY_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CLASS_KEY_CPL_6

*Table 733. km_csr.OTP_CLASS_KEY_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_CLASS_KEY_CPL_7

*Table 734. km_csr.OTP_CLASS_KEY_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_READ_LOCK

*Table 735. km_csr.OTP_READ_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1S | 0x0 | Lock OTP_SEP_SYS_ID_VAL/CPL reads (returns zero when locked) |
| 7 | `sep_sip_id` | RW1S | 0x0 | Lock OTP_SEP_SIP_ID_VAL/CPL reads (returns zero when locked) |
| 6 | `sep_chiplet_id` | RW1S | 0x0 | Lock OTP_SEP_CHIPLET_ID_VAL/CPL reads (returns zero when locked) |
| 5 | `class_key` | RW1S | 0x0 | Lock OTP_CLASS_KEY_VAL/CPL reads (returns zero when locked) |
| 4 | `sys_uid` | RW1S | 0x0 | Lock OTP_SYS_UID_VAL/CPL reads (returns zero when locked) |
| 3 | `sip_uid` | RW1S | 0x0 | Lock OTP_SIP_UID_VAL/CPL reads (returns zero when locked) |
| 2 | `chiplet_uid` | RW1S | 0x0 | Lock OTP_CHIPLET_UID_VAL/CPL reads (returns zero when locked) |
| 1 | `demotion` | RW1S | 0x0 | Lock OTP_DEMOTION_STATE reads (returns zero when locked) |
| 0 | `life_cycle` | RW1S | 0x0 | Lock OTP_LIFE_CYCLE reads (returns zero when locked) |

### OTP_CHANGE_STATUS

*Table 736. km_csr.OTP_CHANGE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1C | 0x0 | OTP_SEP_SYS_ID value changed since last cycle. Sticky, write 1 to clear. |
| 7 | `sep_sip_id` | RW1C | 0x0 | OTP_SEP_SIP_ID value changed since last cycle. Sticky, write 1 to clear. |
| 6 | `sep_chiplet_id` | RW1C | 0x0 | OTP_SEP_CHIPLET_ID value changed since last cycle. Sticky, write 1 to clear. |
| 5 | `class_key` | RW1C | 0x0 | OTP_CLASS_KEY value changed since last cycle. Sticky, write 1 to clear. |
| 4 | `sys_uid` | RW1C | 0x0 | OTP_SYS_UID value changed since last cycle. Sticky, write 1 to clear. |
| 3 | `sip_uid` | RW1C | 0x0 | OTP_SIP_UID value changed since last cycle. Sticky, write 1 to clear. |
| 2 | `chiplet_uid` | RW1C | 0x0 | OTP_CHIPLET_UID value changed since last cycle. Sticky, write 1 to clear. |
| 1 | `demotion` | RW1C | 0x0 | OTP_DEMOTION_STATE value changed since last cycle. Sticky, write 1 to clear. |
| 0 | `life_cycle` | RW1C | 0x0 | OTP_LIFE_CYCLE value changed since last cycle. Sticky, write 1 to clear. |

### OTP_READ_LOCK_COLD

*Table 737. km_csr.OTP_READ_LOCK_COLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `rsvd` | R | 0x0 | Reserved |
| 8 | `sep_sys_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_SYS_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_sys_id \| OTP_READ_LOCK_COLD.sep_sys_id. |
| 7 | `sep_sip_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_SIP_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_sip_id \| OTP_READ_LOCK_COLD.sep_sip_id. |
| 6 | `sep_chiplet_id` | RW1S | 0x0 | Cold-reset read-lock for OTP_SEP_CHIPLET_ID_VAL/CPL. Effective lock = OTP_READ_LOCK.sep_chiplet_id \| OTP_READ_LOCK_COLD.sep_chiplet_id. |
| 5 | `class_key` | RW1S | 0x0 | Cold-reset read-lock for OTP_CLASS_KEY_VAL/CPL. Effective lock = OTP_READ_LOCK.class_key \| OTP_READ_LOCK_COLD.class_key. |
| 4 | `sys_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_SYS_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.sys_uid \| OTP_READ_LOCK_COLD.sys_uid. |
| 3 | `sip_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_SIP_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.sip_uid \| OTP_READ_LOCK_COLD.sip_uid. |
| 2 | `chiplet_uid` | RW1S | 0x0 | Cold-reset read-lock for OTP_CHIPLET_UID_VAL/CPL. Effective lock = OTP_READ_LOCK.chiplet_uid \| OTP_READ_LOCK_COLD.chiplet_uid. |
| 1 | `demotion` | RW1S | 0x0 | Cold-reset read-lock for OTP_DEMOTION_STATE. Effective lock = OTP_READ_LOCK.demotion \| OTP_READ_LOCK_COLD.demotion. |
| 0 | `life_cycle` | RW1S | 0x0 | Cold-reset read-lock for OTP_LIFE_CYCLE. Effective lock = OTP_READ_LOCK.life_cycle \| OTP_READ_LOCK_COLD.life_cycle. |

### OTP_SEP_CHIPLET_ID_VAL_0

*Table 738. km_csr.OTP_SEP_CHIPLET_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_1

*Table 739. km_csr.OTP_SEP_CHIPLET_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_2

*Table 740. km_csr.OTP_SEP_CHIPLET_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_3

*Table 741. km_csr.OTP_SEP_CHIPLET_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_4

*Table 742. km_csr.OTP_SEP_CHIPLET_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_5

*Table 743. km_csr.OTP_SEP_CHIPLET_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_6

*Table 744. km_csr.OTP_SEP_CHIPLET_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_VAL_7

*Table 745. km_csr.OTP_SEP_CHIPLET_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_CHIPLET_ID_CPL_0

*Table 746. km_csr.OTP_SEP_CHIPLET_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_1

*Table 747. km_csr.OTP_SEP_CHIPLET_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_2

*Table 748. km_csr.OTP_SEP_CHIPLET_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_3

*Table 749. km_csr.OTP_SEP_CHIPLET_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_4

*Table 750. km_csr.OTP_SEP_CHIPLET_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_5

*Table 751. km_csr.OTP_SEP_CHIPLET_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_6

*Table 752. km_csr.OTP_SEP_CHIPLET_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_CHIPLET_ID_CPL_7

*Table 753. km_csr.OTP_SEP_CHIPLET_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SIP_ID_VAL_0

*Table 754. km_csr.OTP_SEP_SIP_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_1

*Table 755. km_csr.OTP_SEP_SIP_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_2

*Table 756. km_csr.OTP_SEP_SIP_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_3

*Table 757. km_csr.OTP_SEP_SIP_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_4

*Table 758. km_csr.OTP_SEP_SIP_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_5

*Table 759. km_csr.OTP_SEP_SIP_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_6

*Table 760. km_csr.OTP_SEP_SIP_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_VAL_7

*Table 761. km_csr.OTP_SEP_SIP_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_0

*Table 762. km_csr.OTP_SEP_SIP_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_1

*Table 763. km_csr.OTP_SEP_SIP_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_2

*Table 764. km_csr.OTP_SEP_SIP_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_3

*Table 765. km_csr.OTP_SEP_SIP_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_4

*Table 766. km_csr.OTP_SEP_SIP_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_5

*Table 767. km_csr.OTP_SEP_SIP_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_6

*Table 768. km_csr.OTP_SEP_SIP_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SIP_ID_CPL_7

*Table 769. km_csr.OTP_SEP_SIP_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_0

*Table 770. km_csr.OTP_SEP_SYS_ID_VAL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_1

*Table 771. km_csr.OTP_SEP_SYS_ID_VAL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_2

*Table 772. km_csr.OTP_SEP_SYS_ID_VAL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_3

*Table 773. km_csr.OTP_SEP_SYS_ID_VAL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_4

*Table 774. km_csr.OTP_SEP_SYS_ID_VAL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_5

*Table 775. km_csr.OTP_SEP_SYS_ID_VAL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### OTP_SEP_SYS_ID_VAL_6

*Table 776. km_csr.OTP_SEP_SYS_ID_VAL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_VAL_7

*Table 777. km_csr.OTP_SEP_SYS_ID_VAL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_0

*Table 778. km_csr.OTP_SEP_SYS_ID_CPL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_1

*Table 779. km_csr.OTP_SEP_SYS_ID_CPL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_2

*Table 780. km_csr.OTP_SEP_SYS_ID_CPL_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_3

*Table 781. km_csr.OTP_SEP_SYS_ID_CPL_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_4

*Table 782. km_csr.OTP_SEP_SYS_ID_CPL_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_5

*Table 783. km_csr.OTP_SEP_SYS_ID_CPL_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_6

*Table 784. km_csr.OTP_SEP_SYS_ID_CPL_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

#### OTP_SEP_SYS_ID_CPL_7

*Table 785. km_csr.OTP_SEP_SYS_ID_CPL_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | - | 32-bit OTP word. No reset; power-up value undefined. |

### Key and Policy Vault Registers

#### Key and Policy Vault (KM Port)

Key entry storage and KM control registers. Access conditioned by lock_write/lock_use.

**NOTE**

**Register Arrays:**

- **KEY_ENTRY[64].WORD[16]**: 1024 registers, base 0x0 (KEY_ENTRY x64 stride 0x40, WORD x16 stride 0x4)
- **CTRL[64]**: 64 registers, base 0x1000, stride 0x4

*Table 786. Key and Policy Vault (KM Port) register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xFFC | KEY_ENTRY[64].WORD[16] | RW | One 32-bit word of a key entry (512-bit key = 16 words). No reset; powers up random. |
| 0x1000 - 0x10FC | CTRL[64] | RW | Per-slot control: lock bits (W1S) and the erase trigger |
| 0x1100 | KPV_SCRAMBLER_KEY | RW | 32-bit key for KPV key entry scrambling. No reset; powers up random. Locked when KPV_SCRAMBLER_CTRL.LOCK=1. |
| 0x1104 | KPV_SCRAMBLER_CTRL | RW | KPV scrambler enable and lock. Reset to 0 on KM reset. LOCK is write-one-only. |

#### KEY_ENTRY[64].WORD[16]

*Table 787. km_kpv.KEY_ENTRY[64].WORD[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | - | 32-bit key data word. No reset for security; power-up value undefined. Actual storage is in km_kpv_regfile; CSR field is a protocol placeholder. |

#### CTRL[64]

*Table 788. km_kpv.CTRL[64] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `rsvd_31_4` | R | 0x0 | Reserved |
| 3 | `seal` | RW1S | 0x0 | Seals this key entry until warm reset. Read-any, write-1-only. Setting this bit also sets lock_write in hardware, so a single write seals the slot: its data can be read but not overwritten. Erase remains available, and erasing a sealed slot retires it rather than freeing it (see ERASE), so a sealed slot's material can be destroyed but its slot never reused. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | `erase` | RW1S | 0x0 | Write-1 to start hardware erase of this slot: an LFSR fills all 16 key words (through the KPV scrambler). Hardware self-clears this bit when the erase completes. Never blocked: neither lock_write, lock_use nor seal prevents an erase. The slot's seal state when the erase completes decides the outcome. Unsealed, the slot CTRL register is cleared and the slot is reusable. Sealed, lock_write stays set and lock_use is set, retiring the slot: its data is destroyed and it can be neither read, rewritten nor reused until warm reset. A seal taken while an erase is already in flight therefore still retires the slot. |
| 1 | `lock_use` | RW1S | 0x0 | Prevents KM read of this key entry data until reset. Read-any, write-1-only. Hardware sets this bit when an erase of a sealed slot completes, retiring the slot: its destroyed contents cannot be read back. |
| 0 | `lock_write` | RW1S | 0x0 | Prevents KM write to this key entry data until reset; this register stays writable so that a locked slot can still be read-locked and erased. Read-any, write-1-only. Hardware holds this bit set for as long as SEAL is set, so sealing a slot write-locks it without a separate write here. |

#### KPV_SCRAMBLER_KEY

*Table 789. km_kpv.KPV_SCRAMBLER_KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `key` | RW | - | Scrambler key. No reset for security. HW clears to 0 on wipe via hwclr. |

#### KPV_SCRAMBLER_CTRL

*Table 790. km_kpv.KPV_SCRAMBLER_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `rsvd` | R | 0x0 | Reserved |
| 1 | `lock` | RW1S | 0x0 | Write-one-only. When 1, key and ENABLE cannot be modified until KM reset. |
| 0 | `enable` | RW | 0x0 | 1 = scramble key entry data on write, descramble on read; 0 = passthrough. |

### DRBG Sampler Registers

### DRBG Sampler

DRBG random data interface and configuration/status registers

*Table 791. DRBG Sampler register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | DATA | R | One word of random data. Read triggers request or returns prefetch. Writes return SLVERR. No reset. |
| 0x4 | CFG | RW | Configuration: PREFETCH enable, TIMEOUT cycles for active CPU read. |
| 0x8 | STATUS | RW | Status: DRBG_READY, PREFETCHED, TIMEOUT_ERR (W1C), STREAM_ERR (W1C), COUNT_BAD, COUNT_GOOD. |
| 0xC | PREFETCH_DATA | R | Prefetched word (read-only, for debugging). Cleared when PREFETCH=0. No reset; power-up undefined. |

#### DATA

*Table 792. km_drbg_sampler.DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | 32-bit random data from DRBG. No reset for security; power-up value undefined. |

#### CFG

*Table 793. km_drbg_sampler.CFG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `timeout` | RW | 0x100 | Cycles to wait for DRBG on active CPU read; 0 = disable. Default 256. |
| 15:1 | `rsvd` | RW | 0x0 | Reserved |
| 0 | `prefetch` | RW | 0x0 | Enable prefetch. When 0, prefetch data register is cleared. |

#### STATUS

*Table 794. km_drbg_sampler.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `count_good` | RW | 0x0 | Counts only reads completed through ST_RESPOND, excluding prefetched words. Saturates at 0xFFFF. Writing any non-zero value clears the field to 0; writing 0 has no effect. |
| 15:8 | `count_bad` | RW | 0x0 | Failed transactions (timeout or error). Saturates at 0xFF. Writing any non-zero value clears the field to 0; writing 0 has no effect. |
| 7:4 | `rsvd` | R | 0x0 | Reserved |
| 3 | `stream_err` | RW1C | 0x0 | Set when there is an error on the DRBG AXI-Stream interface (e.g. TVALID deasserted before TREADY). Write 1 to clear. |
| 2 | `timeout_err` | RW1C | 0x0 | Set when a DRBG read from the KM CPU times out. Write 1 to clear. |
| 1 | `prefetched` | R | 0x0 | Prefetched data available |
| 0 | `drbg_ready` | R | 0x0 | TVALID asserted from DRBG |

#### PREFETCH_DATA

*Table 795. km_drbg_sampler.PREFETCH_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | Prefetched 32-bit random data. No reset for security; power-up value undefined. |

### KM Mailbox Registers

### Key Manager Mailbox KM Side

Register interface for KM CPU to access mailbox (write to outbound FIFO, read from inbound FIFO)

*Table 796. Key Manager Mailbox KM Side register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | KM_WRITE_DATA | W | Write data to outbound FIFO (KM→SEP messages) |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x4 | KM_WRITE_SEPARATOR | RW | Write 1 to set message separator on next outbound (KM→SEP) write. Cleared by hardware when that write completes. |
| 0x8 | KM_READ_DATA | R | Read data from inbound FIFO (SEP→KM messages) |
| 0xC | KM_STATUS | RW | FIFO status information |
| 0x10 | KM_IRQ_STATUS | RW | Interrupt status register |
| 0x14 | KM_IRQ_ENABLE | RW | Interrupt enable register for KM CPU |
| 0x18 | KM_CTRL | RW | Control register for mailbox behavior configuration |

### KM_WRITE_DATA

*Table 797. km_mailbox_km.KM_WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | 0x0 | 32-bit data word to write to outbound FIFO |

### KM_WRITE_SEPARATOR

*Table 798. km_mailbox_km.KM_WRITE_SEPARATOR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `set` | RW | 0x0 | Write 1 to set message separator on next outbound write. Cleared by hardware when that write completes. |

### KM_READ_DATA

*Table 799. km_mailbox_km.KM_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | 0x0 | 32-bit data word read from inbound FIFO |

### KM_STATUS

*Table 800. km_mailbox_km.KM_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:26 | `rsvd` | R | 0x0 | Reserved |
| 25 | `outbound_separator` | R | 0x0 | Message separator: 1 if the last word read from outbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 24 | `inbound_separator` | R | 0x0 | Message separator: 1 if the last word read from inbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 23 | `outbound_underflow` | RW1C | 0x0 | Outbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |
| 22 | `inbound_underflow` | RW1C | 0x0 | Inbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |
| 21 | `outbound_overflow` | RW1C | 0x0 | Outbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 20 | `inbound_overflow` | RW1C | 0x0 | Inbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 19:12 | `outbound_depth` | R | 0x0 | Outbound FIFO fill level (number of words in FIFO) |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 11:4 | `inbound_depth` | R | 0x0 | Inbound FIFO fill level (number of words in FIFO) |
| 3 | `outbound_full` | R | 0x0 | Outbound FIFO is full (cannot write more data) |
| 2 | `outbound_empty` | R | 0x1 | Outbound FIFO is empty (no data available for SEP to read) |
| 1 | `inbound_full` | R | 0x0 | Inbound FIFO is full |
| 0 | `inbound_empty` | R | 0x1 | Inbound FIFO is empty (no data available for KM to read) |

### KM_IRQ_STATUS

*Table 801. km_mailbox_km.KM_IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_sep` | RW1C | 0x0 | SEP performed a mailbox flush. Sticky, write 1 to clear. |
| 3 | `inbound_underflow` | RW1C | 0x0 | Inbound FIFO underflow detected. Sticky, write 1 to clear. |
| 2 | `outbound_overflow` | RW1C | 0x0 | Outbound FIFO overflow detected. Sticky, write 1 to clear. |
| 1 | `outbound_write_space_avail` | R | 0x0 | Outbound FIFO has space available for KM to write (level-sensitive). 1 = space available, 0 = full. |
| 0 | `inbound_read_data_avail` | R | 0x0 | Inbound FIFO has data available for KM to read (level-sensitive). 1 = data available, 0 = empty. |

### KM_IRQ_ENABLE

*Table 802. km_mailbox_km.KM_IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_sep_en` | RW | 0x0 | Enable interrupt to KM when SEP performs a mailbox flush |
| 3 | `inbound_underflow_en` | RW | 0x0 | Enable interrupt to KM when inbound FIFO underflow is detected |
| 2 | `outbound_overflow_en` | RW | 0x0 | Enable interrupt to KM when outbound FIFO overflow is detected |
| 1 | `outbound_write_space_avail_en` | RW | 0x0 | Enable interrupt to KM when outbound FIFO has space available (KM→SEP messages) |
| 0 | `inbound_read_data_avail_en` | RW | 0x0 | Enable interrupt to KM when inbound FIFO has data available (SEP→KM messages) |

### KM_CTRL

*Table 803. km_mailbox_km.KM_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `rsvd` | R | 0x0 | Reserved |
| 2 | `flush` | RW | 0x0 | Write 1 to flush all mailbox FIFOs (inbound and outbound). Cleared by hardware when flush completes. |
| 1 | `inbound_underflow_resp` | RW | 0x0 | Response type for inbound FIFO underflow: 0=SLVERR (default), 1=OKAY |
| 0 | `outbound_overflow_resp` | RW | 0x0 | Response type for outbound FIFO overflow: 0=SLVERR (default), 1=OKAY |

### SEP Mailbox Registers

### Key Manager Mailbox SEP Side

Register interface for SEP host to access mailbox (write to inbound FIFO, read from outbound FIFO)

*Table 804. Key Manager Mailbox SEP Side register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | SEP_WRITE_DATA | W | Write data to inbound FIFO (SEP→KM messages) |
| 0x4 | SEP_WRITE_SEPARATOR | RW | Write 1 to set message separator on next inbound (SEP→KM) write. Cleared by hardware when that write completes. |
| 0x8 | SEP_READ_DATA | R | Read data from outbound FIFO (KM→SEP messages) |
| 0xC | SEP_STATUS | RW | FIFO status information |
| 0x10 | SEP_IRQ_STATUS | RW | Interrupt status register |
| 0x14 | SEP_IRQ_ENABLE | RW | Interrupt enable register for SEP host |
| 0x18 | SEP_CTRL | RW | Control register for mailbox behavior configuration |

#### SEP_WRITE_DATA

*Table 805. km_mailbox_sep.SEP_WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | 0x0 | 32-bit data word to write to inbound FIFO |

#### SEP_WRITE_SEPARATOR

*Table 806. km_mailbox_sep.SEP_WRITE_SEPARATOR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `set` | RW | 0x0 | Write 1 to set message separator on next inbound write. Cleared by hardware when that write completes. |

#### SEP_READ_DATA

*Table 807. km_mailbox_sep.SEP_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | 0x0 | 32-bit data word read from outbound FIFO |

#### SEP_STATUS

*Table 808. km_mailbox_sep.SEP_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:26 | `rsvd` | R | 0x0 | Reserved |
| 25 | `outbound_separator` | R | 0x0 | Message separator: 1 if the last word read from outbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 24 | `inbound_separator` | R | 0x0 | Message separator: 1 if the last word read from inbound FIFO had the separator bit set (last word of message). Read-only, not sticky. |
| 23 | `outbound_underflow` | RW1C | 0x0 | Outbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 22 | `inbound_underflow` | RW1C | 0x0 | Inbound FIFO underflow detected (read attempted when empty). Sticky, write 1 to clear. |
| 21 | `outbound_overflow` | RW1C | 0x0 | Outbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 20 | `inbound_overflow` | RW1C | 0x0 | Inbound FIFO overflow detected (write attempted when full). Sticky, write 1 to clear. |
| 19:12 | `outbound_depth` | R | 0x0 | Outbound FIFO fill level (number of words in FIFO) |
| 11:4 | `inbound_depth` | R | 0x0 | Inbound FIFO fill level (number of words in FIFO) |
| 3 | `outbound_full` | R | 0x0 | Outbound FIFO is full |
| 2 | `outbound_empty` | R | 0x1 | Outbound FIFO is empty (no data available for SEP to read) |
| 1 | `inbound_full` | R | 0x0 | Inbound FIFO is full (cannot write more data) |
| 0 | `inbound_empty` | R | 0x1 | Inbound FIFO is empty (no data available for KM to read) |

### SEP_IRQ_STATUS

*Table 809. km_mailbox_sep.SEP_IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_km` | RW1C | 0x0 | KM performed a mailbox flush. Sticky, write 1 to clear. |
| 3 | `outbound_underflow` | RW1C | 0x0 | Outbound FIFO underflow detected. Sticky, write 1 to clear. |
| 2 | `inbound_overflow` | RW1C | 0x0 | Inbound FIFO overflow detected. Sticky, write 1 to clear. |
| 1 | `inbound_write_space_avail` | R | 0x0 | Inbound FIFO has space available for SEP to write (level-sensitive). 1 = space available, 0 = full. |
| 0 | `outbound_read_data_avail` | R | 0x0 | Outbound FIFO has data available for SEP to read (level-sensitive). 1 = data available, 0 = empty. |

### SEP_IRQ_ENABLE

*Table 810. km_mailbox_sep.SEP_IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `rsvd` | R | 0x0 | Reserved |
| 4 | `flushed_by_km_en` | RW | 0x0 | Enable interrupt to SEP when KM performs a mailbox flush |
| 3 | `outbound_underflow_en` | RW | 0x0 | Enable interrupt to SEP when outbound FIFO underflow is detected |
| 2 | `inbound_overflow_en` | RW | 0x0 | Enable interrupt to SEP when inbound FIFO overflow is detected |
| 1 | `inbound_write_space_avail_en` | RW | 0x0 | Enable interrupt to SEP when inbound FIFO has space available (SEP→KM messages) |
| 0 | `outbound_read_data_avail_en` | RW | 0x0 | Enable interrupt to SEP when outbound FIFO has data available (KM→SEP messages) |

### SEP_CTRL

*Table 811. km_mailbox_sep.SEP_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `rsvd` | R | 0x0 | Reserved |
| 2 | `flush` | RW | 0x0 | Write 1 to flush all mailbox FIFOs (inbound and outbound). Cleared by hardware when flush completes. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `outbound_underflow_resp` | RW | 0x0 | Response type for outbound FIFO underflow: 0=SLVERR (default), 1=OKAY |
| 0 | `inbound_overflow_resp` | RW | 0x0 | Response type for inbound FIFO overflow: 0=SLVERR (default), 1=OKAY |

### Adams Bridge Key Interface Registers

### Adams Bridge Key Sideload Registers

Single AXI4-Lite window for all KM sideload interfaces to Adams Bridge. Contains four dual-share secret-seed inputs and one shared-key output.

**NOTE**

**Register Arrays:**

- **MLDSA_SEED.KEY_SHARE0[8]**: 8 registers, base 0x0, stride 0x4
- **MLDSA_SEED.KEY_SHARE1[8]**: 8 registers, base 0x20, stride 0x4
- **MLKEM_SEED_D.KEY_SHARE0[8]**: 8 registers, base 0x100, stride 0x4
- **MLKEM_SEED_D.KEY_SHARE1[8]**: 8 registers, base 0x120, stride 0x4
- **MLKEM_SEED_Z.KEY_SHARE0[8]**: 8 registers, base 0x200, stride 0x4
- **MLKEM_SEED_Z.KEY_SHARE1[8]**: 8 registers, base 0x220, stride 0x4
- **MLKEM_MSG.KEY_SHARE0[8]**: 8 registers, base 0x300, stride 0x4
- **MLKEM_MSG.KEY_SHARE1[8]**: 8 registers, base 0x320, stride 0x4
- **KEY[8]**: 8 registers, base 0x400, stride 0x4

*Table 812. Adams Bridge Key Sideload Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | MLDSA_SEED.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x20 - 0x3C | MLDSA_SEED.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x40 | MLDSA_SEED.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x100 - 0x11C | MLKEM_SEED_D.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x120 - 0x13C | MLKEM_SEED_D.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x140 | MLKEM_SEED_D.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x200 - 0x21C | MLKEM_SEED_Z.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x220 - 0x23C | MLKEM_SEED_Z.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x240 | MLKEM_SEED_Z.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x300 - 0x31C | MLKEM_MSG.KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x320 - 0x33C | MLKEM_MSG.KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x340 | MLKEM_MSG.KEY_CTRL | RW | Seed control register. Writing KEY_VALID=1 marks the seed as loaded. Write 0 to invalidate. The KV shim gates the AB kv_read response on this bit. |
| 0x400 - 0x41C | KEY[8] | R | One 32-bit word of ML-KEM shared key output. Written by hardware when key is ready; read by KM firmware to consume. Hardware clears all words to 0 when KEY_CTRL.KEY_VALID is cleared (via hwclr). |
| 0x420 | MLKEM_SHARED_KEY.KEY_CTRL | RW | Shared-key control. HW sets KEY_VALID when a new key is ready. Firmware writes 0 to clear (consume and zeroize). |
| 0x424 | IRQ_STATUS | RW | Interrupt status for ML-KEM shared key. Sticky bit set by hardware when a new shared key becomes valid; cleared by writing 1. |
| 0x428 | IRQ_ENABLE | RW | Interrupt enable for ML-KEM shared key. Mask for the PicoRV32 IRQ vector bit driven by the shared-key-valid event. |

### MLDSA_SEED.KEY_SHARE0[8]

*Table 813. abr_wrapper_key.MLDSA_SEED.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### MLDSA_SEED.KEY_SHARE1[8]

*Table 814. abr_wrapper_key.MLDSA_SEED.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### MLDSA_SEED.KEY_CTRL

*Table 815. abr_wrapper_key.MLDSA_SEED.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

### MLKEM_SEED_D.KEY_SHARE0[8]

*Table 816. abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

### MLKEM_SEED_D.KEY_SHARE1[8]

*Table 817. abr_wrapper_key.MLKEM_SEED_D.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### MLKEM_SEED_D.KEY_CTRL

*Table 818. abr_wrapper_key.MLKEM_SEED_D.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

#### MLKEM_SEED_Z.KEY_SHARE0[8]

*Table 819. abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### MLKEM_SEED_Z.KEY_SHARE1[8]

*Table 820. abr_wrapper_key.MLKEM_SEED_Z.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### MLKEM_SEED_Z.KEY_CTRL

*Table 821. abr_wrapper_key.MLKEM_SEED_Z.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

#### MLKEM_MSG.KEY_SHARE0[8]

*Table 822. abr_wrapper_key.MLKEM_MSG.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### MLKEM_MSG.KEY_SHARE1[8]

*Table 823. abr_wrapper_key.MLKEM_MSG.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### MLKEM_MSG.KEY_CTRL

*Table 824. abr_wrapper_key.MLKEM_MSG.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, the seed is loaded and the KV shim will serve it to Adams Bridge. Write 0 to invalidate and allow a new seed to be written. |

#### KEY[8]

*Table 825. abr_wrapper_key.KEY[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | 0x0 | 32-bit shared-key word. Hardware writes on key-ready event; hardware clears to 0 on key-consume event (KEY_VALID cleared). |

#### MLKEM_SHARED_KEY.KEY_CTRL

*Table 826. abr_wrapper_key.MLKEM_SHARED_KEY.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW0C | 0x0 | 1 = shared key is valid and ready for firmware consumption. Hardware sets this bit on key-ready event. Write 0 to clear; write 1 has no effect (software cannot set this bit). |

#### IRQ_STATUS

*Table 827. abr_wrapper_key.IRQ_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW1C | 0x0 | ML-KEM shared key valid. Sticky, write 1 to clear. |

#### IRQ_ENABLE

*Table 828. abr_wrapper_key.IRQ_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid_en` | RW | 0x0 | 1 = enable PicoRV32 IRQ when ML-KEM shared key is valid. 0 = mask the interrupt (KEY_CTRL.KEY_VALID can still be polled). |

### AES Key Interface Registers

#### AES Wrapper Key Registers

Key storage registers for the AES sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT AES core.

**NOTE**

**Register Arrays:**

- **KEY_SHARE0[8]**: 8 registers, base 0x0, stride 0x4
- **KEY_SHARE1[8]**: 8 registers, base 0x20, stride 0x4

*Table 829. AES Wrapper Key Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x20 - 0x3C | KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x40 | KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the AES core. |

#### KEY_SHARE0[8]

*Table 830. aes_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_SHARE1[8]

*Table 831. aes_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_CTRL

*Table 832. aes_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### HMAC Key Interface Registers

### HMAC Wrapper Key Registers

Key storage registers for the HMAC sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT HMAC core.

**NOTE**

**Register Arrays:**

- **KEY_SHARE0[8]**: 8 registers, base 0x0, stride 0x4
- **KEY_SHARE1[8]**: 8 registers, base 0x20, stride 0x4

*Table 833. HMAC Wrapper Key Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x20 - 0x3C | KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x40 | KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the HMAC core. |

#### KEY_SHARE0[8]

*Table 834. hmac_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_SHARE1[8]

*Table 835. hmac_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_CTRL

*Table 836. hmac_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### KMAC Key Interface Registers

#### KMAC Wrapper Key Registers

Key storage registers for the KMAC sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT KMAC core.

**NOTE**

**Register Arrays:**

- **KEY_SHARE0[8]**: 8 registers, base 0x0, stride 0x4
- **KEY_SHARE1[8]**: 8 registers, base 0x20, stride 0x4

*Table 837. KMAC Wrapper Key Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | KEY_SHARE0[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x20 - 0x3C | KEY_SHARE1[8] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x40 | KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the KMAC core. |

#### KEY_SHARE0[8]

*Table 838. kmac_wrapper_key.KEY_SHARE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_SHARE1[8]

*Table 839. kmac_wrapper_key.KEY_SHARE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_CTRL

*Table 840. kmac_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### OTBN Key Interface Registers

#### OTBN Wrapper Key Registers

Key storage registers for the OTBN sideload interface. Written by Key Manager CPU, drives keymgr_key_i on the OT OTBN core.

**NOTE**

**Register Arrays:**

- **KEY_SHARE0[12]**: 12 registers, base 0x0, stride 0x4
- **KEY_SHARE1[12]**: 12 registers, base 0x30, stride 0x4

*Table 841. OTBN Wrapper Key Registers register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x2C | KEY_SHARE0[12] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x30 - 0x5C | KEY_SHARE1[12] | W | One 32-bit word of key data. Write-only; reads return 0. No reset; powers up with undefined value. |
| 0x60 | KEY_CTRL | RW | Key control register. Writing KEY_VALID=1 asserts the sideload valid signal to the OTBN core. |

#### KEY_SHARE0[12]

*Table 842. otbn_wrapper_key.KEY_SHARE0[12] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

#### KEY_SHARE1[12]

*Table 843. otbn_wrapper_key.KEY_SHARE1[12] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | W | - | 32-bit key data word. Write-only for security. |

**KEY_CTRL**

*Table 844. otbn_wrapper_key.KEY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd` | R | 0x0 | Reserved |
| 0 | `key_valid` | RW | 0x0 | When 1, asserts keymgr_key_i.valid to indicate key is loaded. Write 0 to invalidate the current key. |

### 2.1.5.12. Memory Map and Register Reference

#### Memory Map

SEP uses 32-bit addressing. The following table outlines the SEP CPU's logical view of the SEP-local address space. External masters use the chiplet-global base programmed in `SEP_GLOBAL_BASE_ADDR`; SEP CPU software uses either direct internal addresses (`0x1000_0000`, `0x1080_0000`, and so on) or the local alias view described below.

*Table 845. SEP CPU Logical Address Space*

| Base Address | End Address | Size | Unit | Description |
|---|---|---|---|---|
| 0x00000000 | 0x0FFFFFFF | 256 MiB | External | External to the chiplet but outside SMU. |
| 0x10000000 | 0x1FFFFFFF | 256 MiB | SEP Local | Direct SEP-local resources. |
| 0x20000000 | 0x3FFFFFFF | 512 MiB | SEP External | Reserved adopter extension aperture. |
| 0x40000000 | 0x7FFFFFFF | 1 GiB | SMC | SMC resources. |
| 0x80000000 | 0xBFFFFFFF | 1 GiB | SMU | SMU resources, including DTP. |
| 0xC0000000 | 0xCFFFFFFF | 256 MiB | SEP CPU Resources | EL2 ICCM, DCCM, and interrupt-controller resources. |
| 0xD0000000 | 0xFFFFFFFF | 768 MiB | SEP Local Alias | CPU-local alias of the direct 0x10000000–0x3FFFFFFF range. |

The `SEP Local Alias` row is an alternate CPU-side name for the same hardware listed in the `SEP Local` and `SEP External` rows. For example, `0xD000_0000` aliases `0x1000_0000` and `0xD080_2000` aliases `0x1080_2000`. This alias range is used by SEP-local masters, while external masters use chiplet-global addresses.

The EL2 CPU resources are mapped into a separate segment than the SEP Local resources due to constraints on the CPU memory map. The ICCM, DCCM, and PIC addresses in the `0xC000_0000` region are statically configured in the CPU core module and are reached through the CPU's internal wiring. The high local-alias mapping covers the `0xD000_0000` and above alias view. CPU CSRs are accessed with dedicated RISC-V instructions and do not appear in the address map.

The high local alias span has a fixed SEP-local size of 768 MiB (`0xD000_0000` through `0xFFFF_FFFF`), corresponding to the direct internal range `0x1000_0000` through `0x3FFF_FFFF`. The chiplet-global SEP aperture configured by `SEP_GLOBAL_BASE_ADDR` / `SEP_REGION_SIZE` controls the SMU xbar and inbound global-to-local remap.

*Table 846. SEP EL2 CPU Resources*

| Base Address | End Address | Size | Unit | Description |
|---|---|---|---|---|
| 0xC0000000 | 0xC003FFFF | 256 KiB | ICCM | Tightly coupled instruction SRAM for the SEP CPU. |
| 0xC0040000 | 0xC005FFFF | 128 KiB | DCCM | Tightly coupled data SRAM for the SEP CPU. |
| 0xC0060000 | 0xC007FFFF | 128 KiB | Reserved | Reserved |
| 0xC0080000 | 0xC0087FFF | 32 KiB | PIC | VeeR EL2 Programmable Interrupt Controller (PIC) memory-mapped registers. Supports up to PIC_TOTAL_INT external interrupt sources with 15 priority levels. |
| 0xC0088000 | 0xCFFFFFFF | 261600 KiB | Reserved | Reserved |

The remaining subsystems are mapped into regions 1 and 2 for the EL2 CPU.

Each row below is the aperture reserved for a unit, and the Decoded Extent column gives the part of it that registers or memory back. The last two columns state how SEP answers a 32-bit access that no register or memory backs, as seen at its inbound AXI port, and Decode Response Codes defines them and their codes.

The TRNG aperture is forwarded in full to an adopter-defined external IP. SEP does not decode offsets inside that window, so every access reaches the external TRNG slave. Intra-aperture decode is then an integrator constraint: when an adopter TRNG is connected, an address that owns no register must return an error and must not alias onto a live register. When no external TRNG is connected, the window is terminated with a DECERR slave, so single-beat reads and writes both return DECERR, refuse the access and never alias onto a live register. See the Integrator Guide.

### Single-Beat Register Access

Register regions support single-beat transactions only. Bursts are supported into ICCM, DCCM, SRAM, the extension port and the outbound SMN windows. The fabric routes a burst on its first address only, so the later beats of a burst to a register region may be answered with errors or alias onto other registers. The crypto region, which spans OTBN through ABR in the generated memory map below, refuses bursts (Single-Beat Access Only).

### Units Unreachable from the Inbound Port

Several SEP units are not connected to the external inbound port and answer `DECERR` to any external access attempt. These units include the Boot ROM, Reset Control, ICCM, DCCM, PIC, and AP/STEE remap regions. The inbound port serves the mailbox and system CSR windows, and forwards all other addresses below 0x4000_0000 (outside the mailbox and system CSR windows) to the SEP local crossbar. Addresses above 0x4000_0000 and the units listed above are not reachable through this external path and receive error responses from the `smn_inbound` port. Their rows in the table below state what the SEP CPU sees.

*Table 847. SEP Component Address Map*

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x10000000 | 0x1003FFFF | 256 KiB | 256 KiB | sep_sram | SEP scratch SRAM. | - | - |
| 0x10040000 | 0x1004FFFF | 64 KiB | 64 KiB | sep_boot_rom | BL0 immutable instruction memory. | - | - |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x10050000 | 0x107FFFFF | 7872 KiB | 7872 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10800000 | 0x10800FFF | 4 KiB | 336 B | DMA Controller | Secure DMA control and status registers. | SLVERR, 0xFFFFFFFF / SLVERR[1] | DECERR, 0xBADCAB1E / DECERR |
| 0x10801000 | 0x10801FFF | 4 KiB | 56 B | Always-On Timer | Wakeup and watchdog timers running on a low-power, always-on clock | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10802000 | 0x1080203F | 64 B | 64 B | SEP Scratch Address Map | Cold-reset scratch register bank. | - | - |
| 0x10802040 | 0x1080207F | 64 B | 64 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10802080 | 0x108020BF | 64 B | 64 B | SEP Scratch Address Map | Warm-reset scratch register bank. | - | - |
| 0x108020C0 | 0x10802FFF | 3904 B | 3904 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10803000 | 0x10803FFF | 4 KiB | 8 B | SEP Reset Control Address Map | Software resets for KM, OTBN, AES, HMAC, KMAC, the internal TRNG, and Adams Bridge. | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10804000 | 0x108FFFFF | 1008 KiB | 1008 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10900000 | 0x1090FFFF | 64 KiB | 48 KiB | OpenTitan Big Number Accelerator | Programmable coprocessor for asymmetric cryptography with SCA and FI countermeasures | SLVERR, 0xFFFFFFFF / SLVERR | DECERR, 0xBADCAB1E / DECERR |
| 0x10910000 | 0x10910FFF | 4 KiB | 140 B | AES Accelerator | AES block-cipher accelerator. | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10911000 | 0x10912FFF | 8 KiB | 8 KiB | HMAC Accelerator | Accelerator for SHA-2 256/384/512-based keyed HMAC and the hash function | SLVERR, 0xFFFFFFFF / SLVERR | - |
| 0x10913000 | 0x10913FFF | 4 KiB | 4 KiB | KMAC Accelerator | Accelerator for Keccak-based keyed hash message authentication code and SHA-3 hash functions; with SCA and FI countermeasures | SLVERR, 0xFFFFFFFF / SLVERR | - |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x10914000 | 0x10914FFF | 4 KiB | 4 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10915000 | 0x109157FF | 2 KiB | 96 B | Cryptographically Secure Random Number Generator | Takes entropy bits to produce cryptographically secure random numbers for consumption by hardware blocks and by software | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10915800 | 0x10915FFF | 2 KiB | 72 B | Entropy Distribution Network | Distributes random numbers produced by CSRNG to hardware blocks | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10916000 | 0x10916FFF | 4 KiB | 380 B | Entropy Source Address Map | Entropy-source configuration, status, and data registers. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| 0x10917000 | 0x10917FFF | 4 KiB | 4 KiB | External TRNG | External TRNG register passthrough; the register layout is adopter-defined | Adopter-defined[2] | - |
| 0x10918000 | 0x1091FFFF | 32 KiB | 24 B | SEP Lifecycle Control Address Map | Lifecycle state and transition control. | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10920000 | 0x10920FFF | 4 KiB | 28 B | Key Manager Mailbox SEP Side | Register interface for SEP host to access mailbox (write to inbound FIFO, read from outbound FIFO) | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10921000 | 0x1092FFFF | 60 KiB | 60 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10930000 | 0x109303FF | 1 KiB | 1 KiB | SEP eFuse Map Address Map | Cached eFuse shadow registers. | - | - |
| 0x10930400 | 0x109304FF | 256 B | 28 B | eFuse Interface Control Address Map | eFuse command and status registers. | - | SLVERR, 0xBADCAB1E / SLVERR |
| 0x10930500 | 0x109305FF | 256 B | 116 B | eFuse Token Registers Address Map | eFuse security-token registers. | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10930600 | 0x1093FFFF | 64000 B | 64000 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10940000 | 0x1094FFFF | 64 KiB | 49176 B | abr | Adams Bridge post-quantum accelerator for ML-DSA and ML-KEM. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x10950000 | 0x1095FFFF | 64 KiB | 24 B | SEP Entropy Pool | Read-only entropy pool FIFO; DATA reads drain conditioned entropy supplied by the native EDN, and writes return SLVERR. | - | SLVERR, 0x0 / SLVERR[3] |
| 0x10960000 | 0x109FFFFF | 640 KiB | 640 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10A00000 | 0x10A0FFFF | 64 KiB | 30800 B | SEP AXI-Lite Mailbox Wrapper Address Map | Eight bidirectional mailbox pairs. | SLVERR, 0x0 / SLVERR | SLVERR, 0x0 / SLVERR |
| 0x10A10000 | 0x10A101FF | 512 B | 512 B | AXI Alias Remap Address Map | TLB Registers for address remapping and attribute tagging | OKAY, 0x0 / OKAY | - |
| 0x10A10200 | 0x10A1027F | 128 B | 128 B | Output Remap Address Map | Address-translation registers that remap outbound transactions and tag their attributes. | - | - |
| 0x10A10280 | 0x10A102FF | 128 B | 128 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10A10300 | 0x10A1037F | 128 B | 128 B | Output Remap Address Map | Address-translation registers that remap outbound transactions and tag their attributes. | - | - |
| 0x10A10380 | 0x10A1FFFF | 64640 B | 64640 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10A20000 | 0x10A203FF | 1 KiB | 1 KiB | AXI Filter Control Address Map | Per-filter access-control registers defining permitted address ranges and transaction attributes. | OKAY, 0x0 / OKAY | - |
| 0x10A20400 | 0x10A20FFF | 3 KiB | 3 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10A21000 | 0x10A211FF | 512 B | 512 B | AXI Filter Control Address Map | Per-filter access-control registers defining permitted address ranges and transaction attributes. | OKAY, 0x0 / OKAY | - |

| Base Address | End Address | Size | Decoded Extent | Unit | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|
| 0x10A21200 | 0x10A2FFFF | 60928 B | 60928 B | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10A30000 | 0x10A31FFF | 8 KiB | 4104 B | SEP CPU Control Address Map | SEP CPU control and interrupt registers. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| 0x10A32000 | 0x10AFFFFF | 824 KiB | 824 KiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x10B00000 | 0x10BFFFFF | 1 MiB | 56 B | SPI Host | External I/O SPI controller bridge. | - | DECERR, 0xBADCAB1E / DECERR |
| 0x10C00000 | 0x10FFFFFF | 4 MiB | 4 MiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x11000000 | 0x117FFFFF | 8 MiB | 8 MiB | ap_region | Address region whose accesses an output remap redirects. | Forwarded | - |
| 0x11800000 | 0x11FFFFFF | 8 MiB | 8 MiB | stee_region | Address region whose accesses an output remap redirects. | Forwarded | - |
| 0x12000000 | 0x1FFFFFFF | 224 MiB | 224 MiB | Reserved | Reserved | DECERR, 0xBADCAB1E / DECERR | - |
| 0x20000000 | 0x3FFFFFFF | 512 MiB | 512 MiB | SEP External | SEP external address map (eFuse SHIM control and execute-in-place window). | Adopter-defined | - |
| 0xC0000000 | 0xC003FFFF | 256 KiB | 256 KiB | sep_iccm | Tightly coupled instruction SRAM for the SEP CPU. | - | - |
| 0xC0040000 | 0xC005FFFF | 128 KiB | 128 KiB | sep_dccm | Tightly coupled data SRAM for the SEP CPU. | - | - |
| 0xC0080000 | 0xC0087FFF | 32 KiB | 21 KiB | EL2_PIC | VeeR EL2 Programmable Interrupt Controller (PIC) memory-mapped registers. Supports up to PIC_TOTAL_INT external interrupt sources with 15 priority levels. | OKAY, 0x0 / OKAY | OKAY, 0x0 / OKAY |

*Notes*

[1] An access to an unmapped offset also latches PIC source 41.
[2] With no TRNG fitted, the reference integration answers DECERR, 0xBADCAB1E / DECERR.
[3] A 32-bit read of the upper word of a register also answers SLVERR, 0x0.

#### Detailed Register Map

The address tables above identify each aperture. Detailed registers are published with the owning block:

See the register references in the CPU, Interrupts, SEP Reset Controller, Life Cycle Controller, OTP Fuse Controller, Watchdog Timer, Cryptographic Subsystem (including AES Cipher, HMAC-SHA2, KMAC-SHA3, OTBN and TRNG and DRBG), DMA, Serial Peripheral Interface, Key Manager and Entropy Source chapters. The shared eFuse IP chapter owns the eFuse MMR reference.

#### External Window Reference Model

The SEP external-window model supplies the eFuse SHIM control register below. The `efuse_shim_ctrl` instance at offset `0x0` fixes where that block sits. `EFUSE_SHIM_SIZE` bytes from that base (`CFG.SEP_EFUSE_SHIM_SIZE` at the SMU, default `0x4`) are routed to the eFuse controller, which forwards them unchanged on `efuse_bank_ctrl` to the adopter's SHIM rather than out the external port. An adopter external map must keep an `efuse_shim_ctrl` instance at that offset. Its execute-in-place region is a memory window rather than a register bank. Adopter-specific external devices require their own register documentation.

#### SEP External

SEP external address map (eFuse SHIM control and execute-in-place window).

*Table 848. SEP External register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | EFUSE_BANK_INIT_TIME | RW | eFuse bank initialization time. Number of cycles to wait for the OTP macro to initialize before sensing can begin. |

#### EFUSE_BANK_INIT_TIME

*Table 849. sep_external.EFUSE_BANK_INIT_TIME fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `init_time` | RW | 0x20 | Most OTP have a initialization time before sensing can be done |

## 2.1.6. System Management Controller (SMC)

The System Management Controller (SMC) performs firmware-driven platform management for each chiplet. Its documentation is organized into focused topic pages covering the subsystem architecture, hardware services, and programming interfaces.

### 2.1.6.1. Overview

The System Management Controller (SMC) is a subsystem in a chiplet that takes on the system management tasks, including managing clock/voltage/reset, configuring the hardware platform, bringing up the system, and reporting system status.

Given its essential role, a SiP with multiple chiplets integrates one SMC per chiplet. Each SMC instance in a chiplet is responsible for system management of the local chiplet. On top of that, a management hierarchy is expected to be built among different SMC instances, according to the OCA hierarchical system architecture, , Sec. 3.1 and 3.2. This means an SMC instance in the primary chiplet is expected to play additional roles to manage SiP level tasks for the secondary chiplets.

The SMC consists of a micro-controller and different facilitative components for various purposes. At high level, SMC's main control logic is implemented in the firmware running on the micro-controller. Hardware facilitative components are added only when certain tasks cannot be implemented by firmware.

Figure 49 shows the SMC microarchitectural diagram.

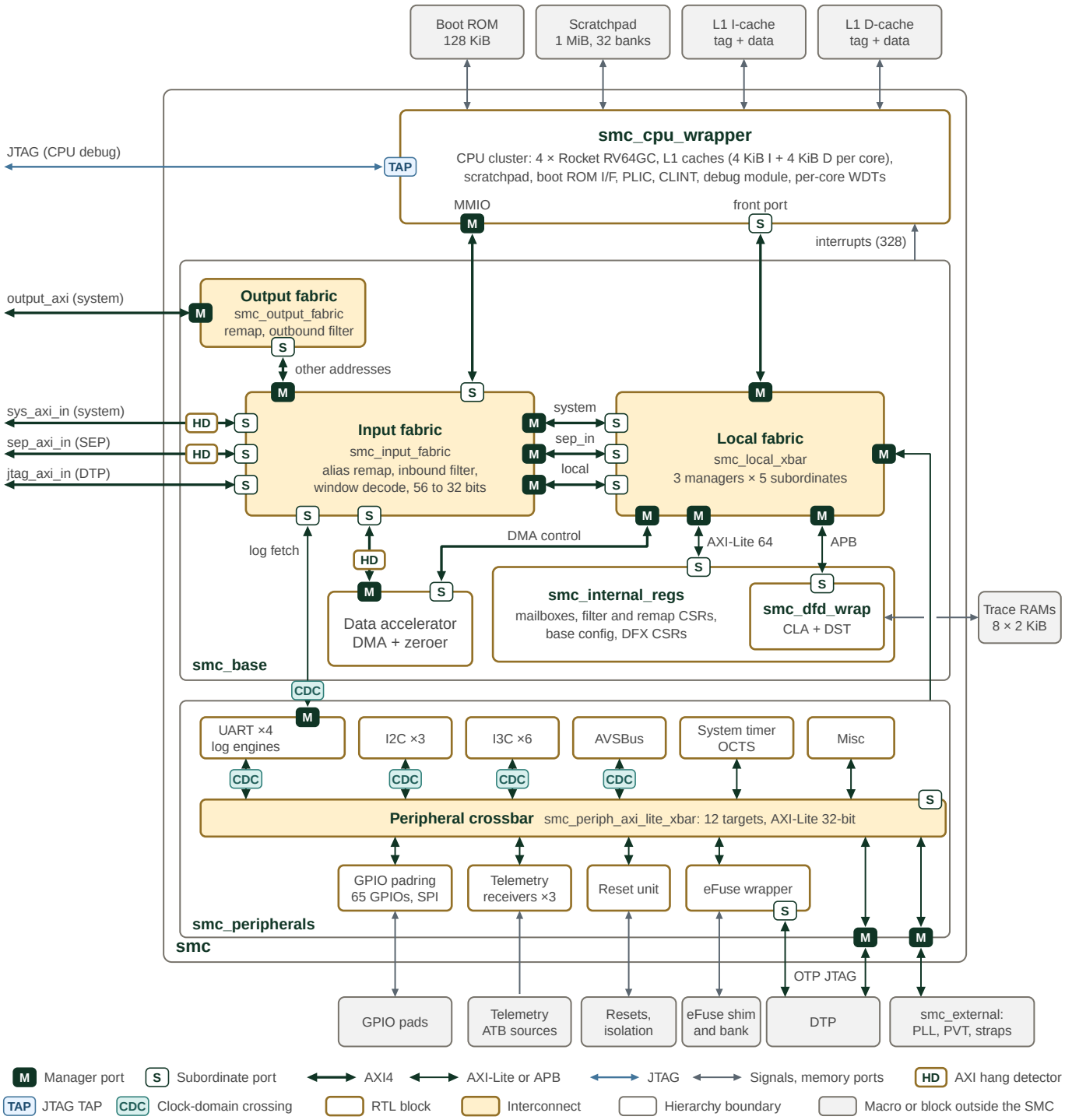


*Figure 49. SMC top-level microarchitecture*

The `smc` module holds three blocks: the CPU cluster `smc_cpu_wrapper`, `smc_base` and `smc_peripherals`. The CPU cluster is drawn as a single block; the CPU and memory system section describes its contents. Every AXI4 and AXI-Lite link carries a manager port (M) at the initiating end and a subordinate port (S) at the other.

The input fabric takes six initiators: the system inbound port, the SEP inbound port, the JTAG debug port from the DTP, the CPU MMIO port, the UART log-engine fetch port and the data accelerator. Traffic inside the SMC window reaches the local fabric on three paths. System traffic passes the inbound filter first, and SEP traffic is not filtered. The CPU MMIO, JTAG, log and data accelerator traffic is alias-remapped and shares the third path. The input fabric forwards other addresses from those four initiators to the output fabric. The output fabric applies the M-mode and Xvisor remaps and the outbound filter, then drives the system outbound port. System and SEP requests outside the SMC window receive a decode error.

The local fabric, `smc_local_xbar`, serves five targets: the CPU front port, the data accelerator control port, the internal CSRs, the DFD registers and the peripheral port. Its internal AXI-Lite crossbar splits the internal CSRs into eight blocks in `smc_internal_regs`: base configuration, the alias, M-mode and Xvisor remaps, the inbound and outbound filters, the mailboxes and the DFX CSRs. `smc_base` also builds the CPU interrupt vector from the external, peripheral, mailbox and internal sources. Its three AXI hang detectors, marked HD on the links they watch, observe the system inbound, SEP inbound and data accelerator AXI links without sitting in the data path; their combined interrupt reaches the PLIC through `smc_peripherals`.

The peripheral crossbar, `smc_periph_axi_lite_xbar`, has twelve targets. Ten of them are inside `smc_peripherals`. The UART, I2C, I3C and AVSBus register paths, and the log-engine fetch path, cross between the SMC clock and the peripheral clock (CDC). The remaining two targets leave the SMC: the DTP control registers and the `smc_external` window. In the reference top `smc_wrapper`, `smc_ip_integration` supplies the grey blocks: the CPU memory macros, the trace RAMs, the eFuse shim and bank model, the GPIO pad shims, and the PLL, PVT and strap register models behind `smc_external`.

The rest of this documentation describes architectural requirements and microarchitectural implementation of SMC's components in OCAH. A consolidated description of OCAH use cases is available in Use Cases.

#### Clocks and Resets

SMC coordinates clock, voltage and reset control. Its processing, reference timing and peripheral interfaces use distinct clock branches. Clocks and Resets describes the domains and reset sequencing.

#### CPU

The RISC-V CPU cluster executes management firmware using its caches and local memories. The CPU chapter describes its configuration, interfaces, isolation, error reporting and debug support.

#### Boot ROM

Boot ROM contains the startup firmware. Scratchpad memory provides runtime storage. The Boot ROM chapter describes the hardware and links to the production firmware manual.

#### Interconnect

AXI4 and AXI4-Lite connect the CPU and management blocks. Remapping, traffic filters and crossbars provide address translation, access control and routing.

#### Interrupts

The PLIC routes system events; CLINT supplies core-local timer and software interrupts. The interrupt chapter describes their integration and handling.

#### Peripherals

DMA performs bulk transfers and Zeroer clears memory. Communication and control services include mailboxes, GPIO, I2C, I3C, UART and AVSBus. Telemetry receivers collect performance data; OCTS supports multi-chiplet timekeeping. PVT sensors provide environmental monitoring through adopter integration.

#### Design-for-Debug and CLA

JTAG, the RISC-V debug module and the Core Logic Analyzer support observation, triggering and trace capture.

#### eFuse Scan Protection

Scan protection controls access to eFuse shadow-register assets. Its dedicated chapter describes the asset classes and access restrictions.

#### Memory Map

The memory map summarizes the address space and points to the block pages for register details.

### 2.1.6.2. Clocks and Resets

SMC separates CPU/fabric operation, peripheral timing, reference-clock reset control and incoming telemetry into distinct clock domains. Reset sources are combined before release is synchronized into each destination domain. Subsystem isolation and reset-retention controls require support in the receiving subsystem.

#### Clock Architecture

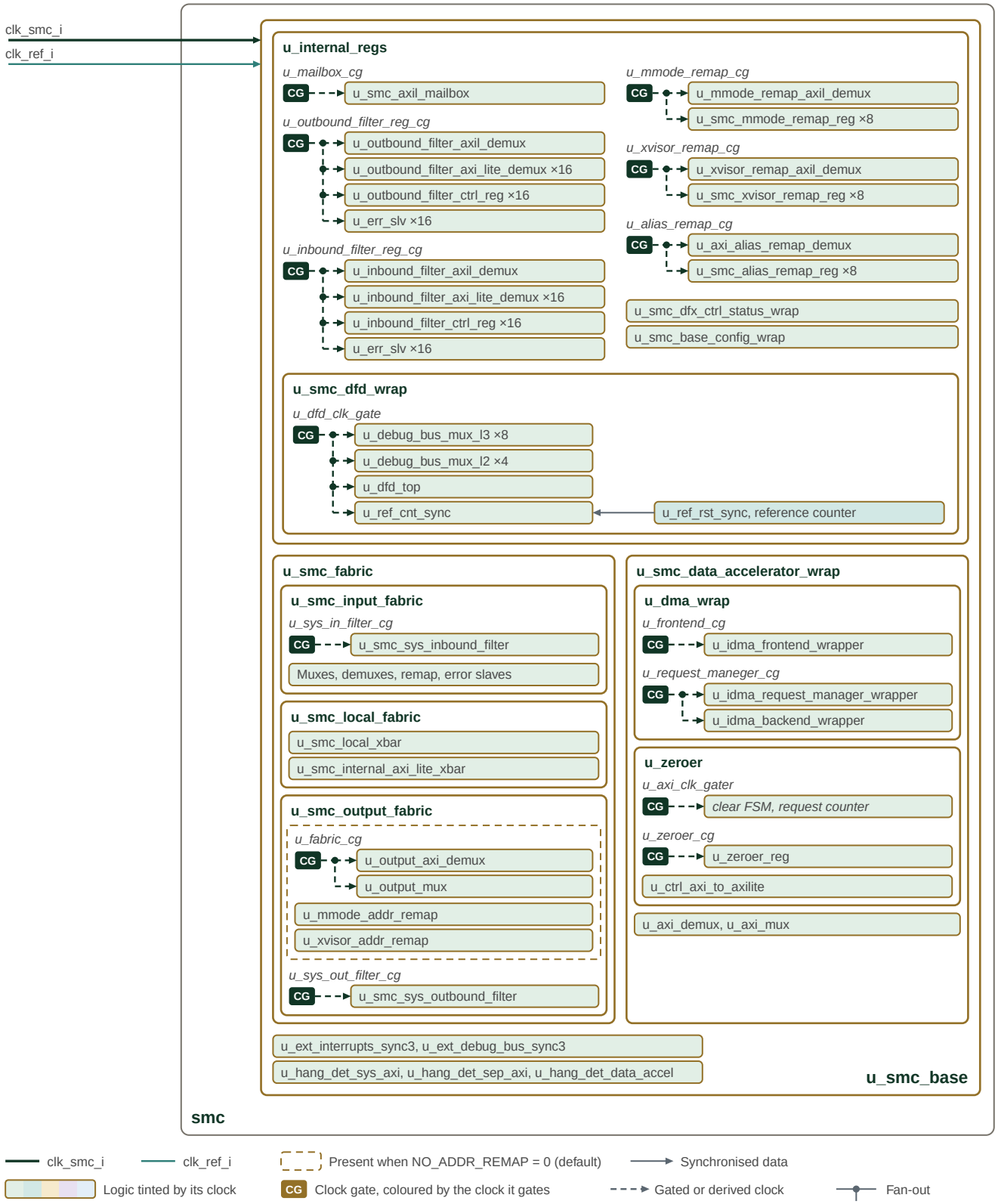


*Figure 50. SMC clock tree and clock domains in u_smc_base*

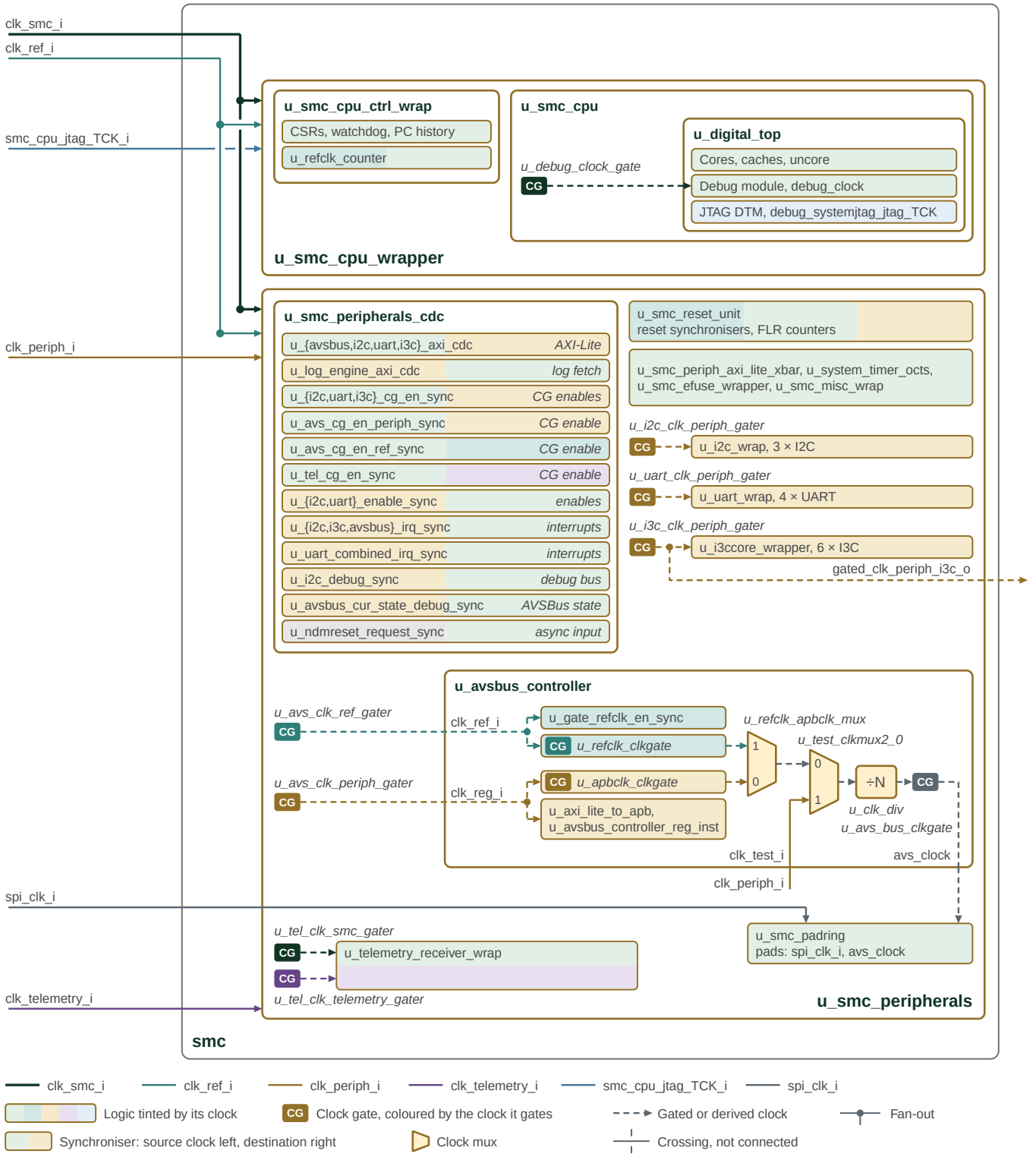


*Figure 51. SMC clock tree and clock domains in u_smc_cpu_wrapper and u_smc_peripherals*

The two figures split the SMC clock tree at its subsystem instances. Each clock input enters on the left in its own colour, with one arrow into each of `u_smc_base`, `u_smc_cpu_wrapper` and `u_smc_peripherals` that it reaches; an arrow into an instance means the clock reaches the logic inside it. Each box is an RTL instance, tinted with the colour of the clock that runs it, and a box split into bands runs on several clocks. A clock gate is drawn as a CG tab coloured by the clock it gates, with the gate's instance name in italics. A dashed arrow runs from the gate to every instance its gated clock reaches, branching at a junction dot. A count such as ×16 is the number of instances a generate loop creates. A box that no gate arrow reaches runs on the ungated clock.

Most gates in u_smc_base are activity-based: each watches the AXI or AXI-Lite port of the logic it clocks and stops the clock once the port has been idle for the programmed hysteresis. Each filter and remap register gate clocks the AXI-Lite demux in front of its register set as well as the registers, and each filter gate also clocks the error slave that answers writes to a locked filter. u_fabric_cg and the instances inside the dashed outline exist only when NO_ADDR_REMAP is 0, the default in smc_config_pkg. u_dfd_clk_gate clocks the debug bus muxes, u_dfd_top and u_ref_cnt_sync, which receives the Gray-coded count of the reference counter beside u_ref_rst_sync and forms the CLA time tick. u_axi_clk_gater clocks logic written directly in u_zeroer rather than an instance.

u_debug_clock_gate runs the debug module clock, the debug_clock port of u_digital_top, while the debugger holds dmactive. The I2C, UART, I3C, AVSBus and telemetry gates are controlled by CLOCK_GATE_CONTROL in u_smc_base. u_smc_peripherals_cdc is drawn as one row per synchroniser or CDC instance, or per group of instances that cross the same way, with the source clock on the left half and the destination clock on the right half. The clock-gate enable synchronisers carry each enable into the ungated clock of the gate it controls. u_ndmreset_request_sync has no source clock: ndmreset_request_i is asynchronous.

Each of the five peripheral enables also steers that peripheral's leg of the register crossbar into an error slave (u_avsbus_access_gate, u_i2c_access_gate, u_uart_access_gate, u_telemetry_access_gate and u_i3c_access_gate in u_smc_peripherals, all on clk_smc_i): while the enable is set, a read of the gated peripheral's window completes with SLVERR and data 0xBADCAB1E and a write completes with SLVERR, instead of waiting on a register block whose clock has stopped. The leg is redirected as soon as the CLOCK_GATE_CONTROL write completes; an access the peripheral has already accepted finishes there, since its clock stops only after the enable has crossed into its domain.

The AVSBus clock starts at u_avs_clk_ref_gater and u_avs_clk_periph_gater, which drive the clk_ref_i and clk_reg_i ports of u_avsbus_controller. Inside it, u_refclk_clkgate and u_apbclk_clkgate feed the glitch-free mux u_refclk_apbclk_mux, selected by avs_clock_select[1]. The test mux u_test_clkmux2_0 takes that output on clk0_i and the test clock input clk_test_i on clk1_i, and selects the test clock in test mode; u_smc_peripherals connects clk_test_i to clk_periph_i. The divider u_clk_div, used while avs_clock_select[0] is set, divides by a programmable value, 4 after reset, and u_avs_bus_clkgate stops the result on idle. The output, avs_clock, leaves through u_smc_padring on GPIO 49. spi_clk_i reaches only the SPI pad datapath in u_smc_padring.

*Table 850. SMC clock-domain responsibilities*

| Clock | Logic and boundary |
|---|---|
| clk_smc_i | CPU cluster, local fabric, remapping/filtering, register crossbar, OCTS, and telemetry decoding/message storage. Individual blocks may gate this clock. |
| clk_ref_i | Power-good/reset conditioning, FLR countdowns, DFD local time tick and peripheral reference-clock inputs. |
| clk_periph_i | AVSBus, I2C, UART/log-engine and I3C operation. Register accesses cross through per-port AXI-Lite CDC bridges after the SMC-domain crossbar. |
| clk_telemetry_i | Incoming ATB side of each telemetry crossing FIFO. The receiver core and its CSRs operate in the SMC domain; the external telemetry reset is rst_telemetry_ni. |

### Timer and Debug Timebases

- OCTS uses clk_smc_i.
- CLINT receives one tick per 1000 CPU-cluster cbus cycles; the uncore input uses the main SMC clock. This timer tick is not the reference-clock input.
- The CPU debug module uses a gated CPU-wrapper clock, enabled by debug activation with a test override. JTAG TCK clocks the scan interface separately.

### Peripheral Timing Requirements

The integrator supplies `clk_periph_i` at a frequency satisfying every enabled peripheral. A minimum frequency of 100 MHz applies; the upper limit is not fixed and is set by the integrator according to timing closure, peripheral IP limits, and system power and performance goals.

The I3C target's worst-case internal SCL-to-SDA turnaround is four core-clock cycles. Meeting the 12 ns target turnaround limit therefore requires a clock period no greater than 3 ns (approximately 333.3 MHz or faster) before allowing for pad and routing delays. The integrator must include those delays in the timing budget and select a higher frequency where needed. See the CHIPS Alliance I3C PHY timing requirements. This is a target-mode requirement; controller-mode timing configuration does not establish target compliance.

### Reset Architecture

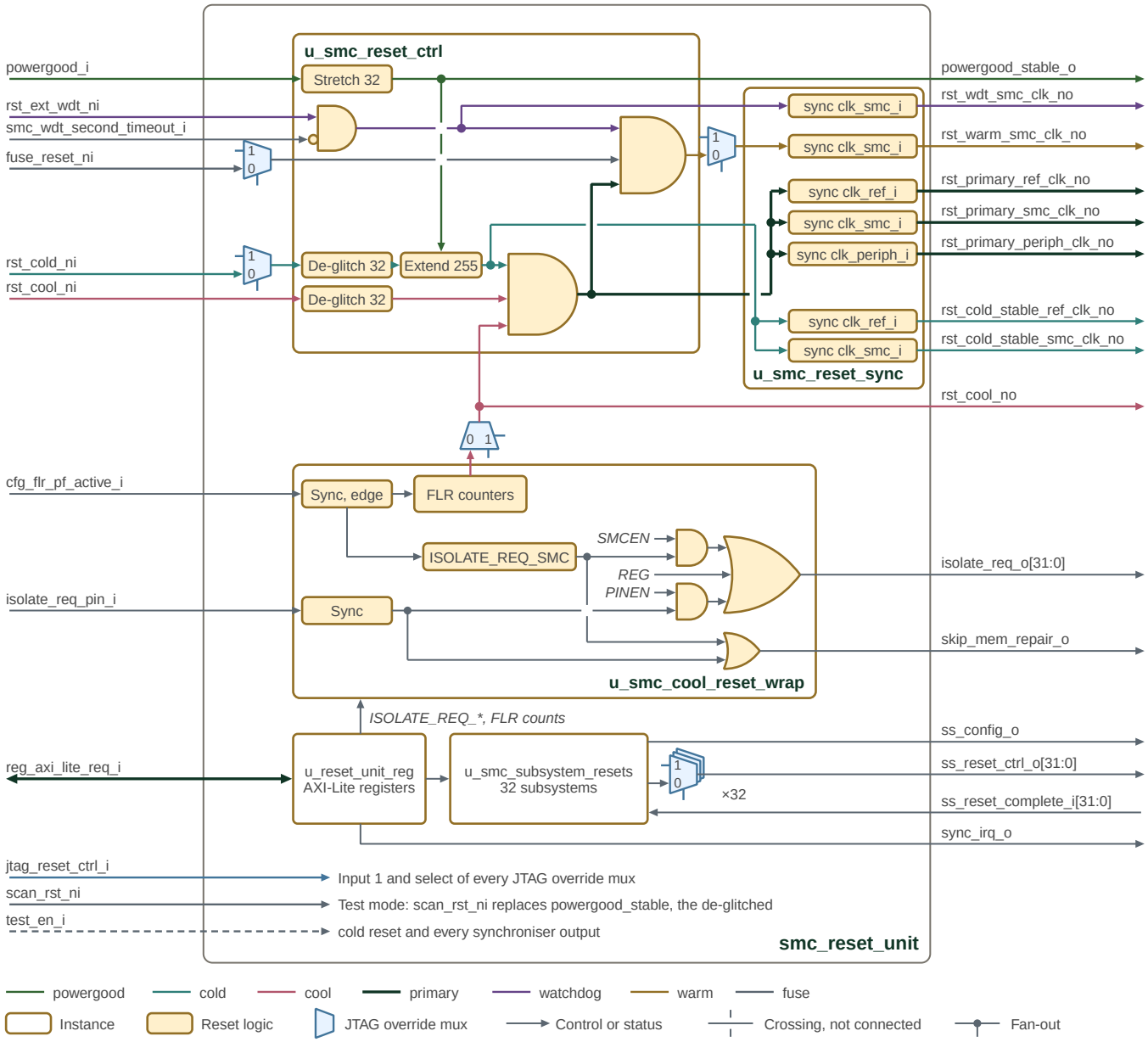


*Figure 52. SMC reset unit architecture*

Reset inputs enter on the left and the unit's outputs leave on the right. Each reset net has its own colour, listed in the

legend. The blue mux symbols are JTAG override muxes: `jtag_reset_ctrl_i` drives their input 1 and select, drawn as short stubs, and each mux passes the value on input 1 while its override bit is set.

In `u_smc_reset_ctrl`, `powergood_i` is stretched by 32 reference clock cycles into `powergood_stable`, which leaves as `powergood_stable_o`, clears the de-glitchers while low and holds the cold-reset extender. The cold reset, after its JTAG override, must stay low for 32 reference clock cycles to pass the de-glitcher, and its release is extended by 255 cycles. The cool-reset pin passes its own 32-cycle de-glitcher. The primary reset is the AND of the stable cold reset, the de-glitched cool-reset pin and the FLR cool reset. The watchdog reset is the AND of `rst_ext_wdt_ni`, which comes from the SEP watchdog, and the inverted SMC watchdog second timeout; `smc_wdt_first_timeout_i` does not reach the reset logic. The warm reset is the AND of the watchdog reset, the primary reset and the fuse reset, which the eFuse wrapper releases once fuse sensing completes and any boot stall ends. Debug-initiated resets are the JTAG overrides of the cold, cool, warm, fuse and per-subsystem resets.

`u_smc_reset_sync` gives each output an asynchronous assertion and a deassertion synchronised to its clock. `u_smc_cool_reset_wrap` runs the FLR sequence and isolation logic described in Function Level Reset and Isolation. Its gates act on each of the 32 bits; the inputs labelled SMCEN, REG and PINEN are the `ISOLATE_REQ_SMCEN_REG`, `ISOLATE_REQ_REG` and `ISOLATE_REQ_PINEN_REG` bits. `skip_mem_repair_o` is high while the synchronised isolation pin or `ISOLATE_REQ_SMC` is set. `u_reset_unit_reg` is reset by `rst_primary_smc_clk_no`, as are the `SS_CONFIG` and `SS_COLD_RESET_N` registers in `u_smc_subsystem_resets`; the isolation registers and FLR counters are reset only by the stable cold reset, so they survive a cool reset. The per-subsystem cold and warm resets leave `ss_reset_ctrl_o` unsynchronised, and each subsystem synchronises them. In test mode, `scan_rst_ni` replaces the stretched powergood, the de-glitched cold reset and every synchroniser output.

#### Power-Good and Cold Reset

`BP_POWERGOOD` is the POR root. A 32-stage reference-clock synchronizer produces `powergood_stable`. Cold reset from `BP_RESETN` is filtered through a 32-bit shift register: once released, assertion requires 32 consecutive low samples. A 255-cycle reference-clock extension holds cold reset after the filtered input releases. Incoming cool reset has the same low-sample filter but no cold-reset extension. These are clock-cycle conditions, not fixed nanosecond delays.

Power-good also enters JTAG/DTP as `pwr_on_rst_ni`. The primary TAP combines it with pad TRST, so loss of power-good resets TAP/TDR state even if TRST is released. Functional cold reset alone does not reset that state.

*Table 851. SMC reset sources and effects*

| Source | Functional effect |
|---|---|
| Power-good low | Holds functional reset and resets the JTAG/TDR POR path. |
| `BP_RESETN` | Filtered and extended cold reset feeds primary reset. |
| Incoming cool reset | Filtered pin request feeds primary reset. |
| FLR cool reset | Reference-domain timed reset feeds primary reset; isolation is controlled separately as described below. |
| External watchdog reset / second SMC watchdog timeout | Assert the watchdog reset term, which also asserts warm reset. The first-timeout input does not enter the reset equation in `smc_reset_ctrl`. |
| Fuse reset | Holds warm reset until released by the fuse-control path. |
| JTAG reset overrides | Select alternate fuse, cold, cool or warm reset values; normal functional equations below assume these overrides and scan mode are inactive. |

#### Primary and Warm Reset

In normal operation the active-low reset equations are:

```
rst_primary_n = stable_cold_rst_n AND stable_cool_rst_n
```

```
                AND rst_cool_from_flr_n
rst_wdt_n     = rst_ext_wdt_ni AND NOT smc_wdt_second_timeout_i
rst_warm_n    = rst_primary_n AND rst_wdt_n AND fuse_reset_ni
```

Primary reset covers the main SMC functional fabric, peripheral control and configuration paths. Warm reset drives CPU core/uncore reset control, including the cluster's interrupt and timer logic, without resetting every primary-domain peripheral. CPU control registers also provide per-core/uncore reset controls; debug reset requests and JTAG overrides should not be treated as one unconditional warm-reset input. See the CPU chapter for its isolation/reset interlock.

#### Reset Synchronization

`smc_reset_sync` uses four-stage `prim_sync_reset` instances for asynchronous assertion and destination-clock-synchronous release. Scan mode selects the scan reset. A stopped destination clock cannot complete functional reset release.

*Table 852. Reset synchronization domains*

| Output from `smc_reset_sync` | Clock | Reset source |
|---|---|---|
| `rst_cold_smc_no` | SMC | Stable cold reset; exported by the unit as `rst_cold_stable_smc_clk_no`. |
| `rst_primary_smc_clk_no` | SMC | Primary reset. |
| `rst_warm_smc_clk_no` | SMC | Warm reset after JTAG override selection. |
| `rst_wdt_smc_clk_no` | SMC | Watchdog reset term. |
| `rst_cold_ref_clk_no` | Reference | Stable cold reset; exported as `rst_cold_stable_ref_clk_no`. |
| `rst_primary_ref_clk_no` | Reference | Primary reset. |
| `rst_primary_periph_clk_no` | Peripheral | Primary reset. |

JTAG overrides for outgoing per-subsystem cold/warm controls leave the unit unsynchronized. The receiving subsystem must synchronize release at its boundary. The separate telemetry input reset belongs to its integration domain.

#### Function Level Reset and Isolation

PCIe `cfg_flr_pf_active_i` crosses independently into the SMC and reference clock domains. Its synchronized rising edge sets the SMC isolation latch and starts the reference-domain reset sequence when enabled by a nonzero hold count. The reset state machine has no isolation-complete input: its delay must be chosen to accommodate the system's isolation requirements.

#### Isolation Controls

Each of the 32 `isolate_req_o` bits is the OR of three requests:

```
ISOLATE_REQ_REG[i]
OR (ISOLATE_REQ_PINEN_REG[i] AND synchronized isolate_req_pin_i)
OR (ISOLATE_REQ_SMCEN_REG[i] AND ISOLATE_REQ_SMC_REG)
```

The FLR edge sets `ISOLATE_REQ_SMC_REG`. After recovery, a write to that register with any write strobe asserted clears the latch; write data is ignored. An FLR edge in the same cycle takes priority. Ending the timed cool reset does not clear the latch or release isolation automatically. Clearing one source does not cancel an isolation request held by either of the others.

The isolation registers and FLR count storage are cold-reset state, so they survive cool reset. `ISOLATE_REQ_PINEN_REG` has a conditional synchronous clear: it clears on cold reset only while the synchronized isolation pin is low. Do not assume an uninitialized pin-enable mask becomes known while that pin is high.

### Timing and Recovery

The FLR count values cross from SMC to the reference domain through handshaked synchronizers; readback in the SMC domain does not acknowledge arrival in the reference domain. `ISOLATE_REQ_FLR_COUNTER_VALUE` selects the pre-reset countdown; zero allows assertion without that countdown. `ISOLATE_REQ_FLR_RESET_COUNTER_VALUE` selects the cool-reset hold count, and must be nonzero to start the timed flow.

On a synchronized FLR edge, the sequencer counts in reference-clock cycles, asserts cool reset, then releases it after the hold countdown. Destination reset synchronizers add their release latency. The sequencer is not a queue of overlapping FLR requests. The FLR preparation and recovery sequence is in the Programmer's Guide.

A zero hold count disables the timed reset sequence, but does not prevent the SMC-domain FLR edge from setting the isolation latch. `ISOLATE_REQ_VIS` reports synchronized pin and cool-reset levels; it is not an isolation-completion handshake.

`skip_mem_repair_o` is the OR of the synchronized isolation pin and the FLR isolation latch. It does not depend on the per-subsystem masks, and a software-only `ISOLATE_REQ_REG` request does not assert it. The receiving logic determines how this request bypasses memory repair/MBIST; it is not a general SRAM-retention guarantee.

### Isolation Connections

Figure 53 shows cool-reset and isolation connections between primary and secondary chiplets. The next three figures repeat the same drawing and highlight the path that each kind of request activates, so they can be compared directly.

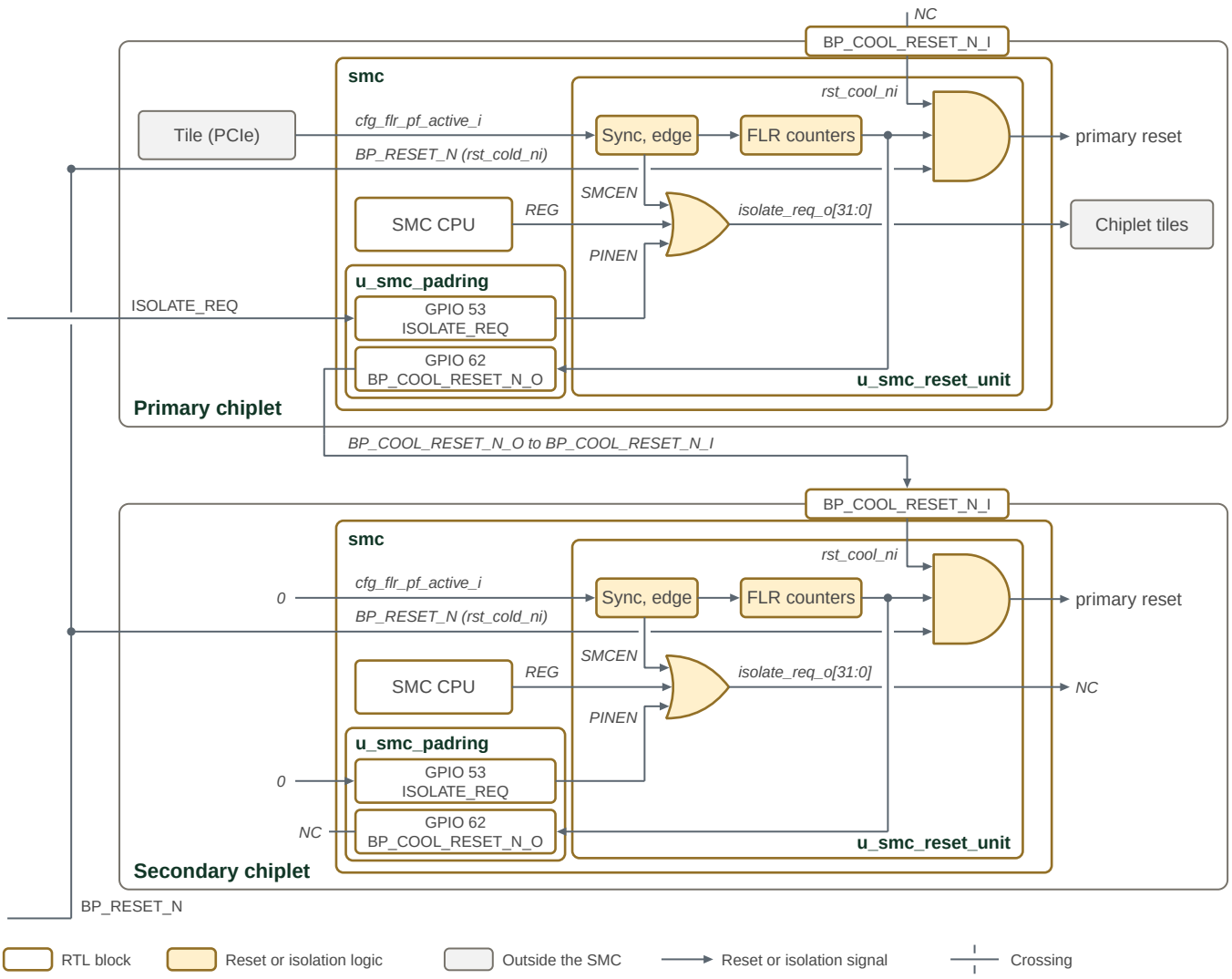


*Figure 53. SMC FLR isolation connections between chiplets*

Each chiplet's SMC takes its cold reset from the board `BP_RESET_N` and its cool reset from its `BP_COOL_RESET_N_I` input; the primary reset is the AND of these and the FLR cool reset. The FLR cool reset also drives the open-drain `BP_COOL_RESET_N_O` pad, GPIO 62, which pulls the line low while the reset is asserted and leaves it undriven otherwise. The primary chiplet's `BP_COOL_RESET_N_O` connects to the secondary chiplet's `BP_COOL_RESET_N_I`, so an FLR on the primary chiplet cool-resets the secondary. To the secondary chiplet, a cold reset on `BP_RESET_N` and a cool reset on `BP_COOL_RESET_N_I` are the same: both assert its primary reset. Only its FLR and isolation registers are reset by the cold reset alone, and the secondary chiplet does not use them. The primary chiplet's `BP_COOL_RESET_N_I` and the secondary chiplet's `BP_COOL_RESET_N_O` are not connected. The board drives `ISOLATE_REQ`, sampled on GPIO 53, into the primary chiplet only; the secondary chiplet ties its `ISOLATE_REQ` and `cfg_flr_pf_active_i` low and leaves its `isolate_req_o` unconnected. The labels REG, PINEN and SMCEN on the isolation OR's inputs name the registers that qualify each source. The drawing leaves out the de-glitchers and synchronisers shown in Figure 52.

Figure 54 shows isolation propagation for an SMC software request. Software sets bits of `ISOLATE_REQ_REG`, which reach `isolate_req_o` directly. Software cannot start the FLR counters, so this request isolates tiles without a cool reset.

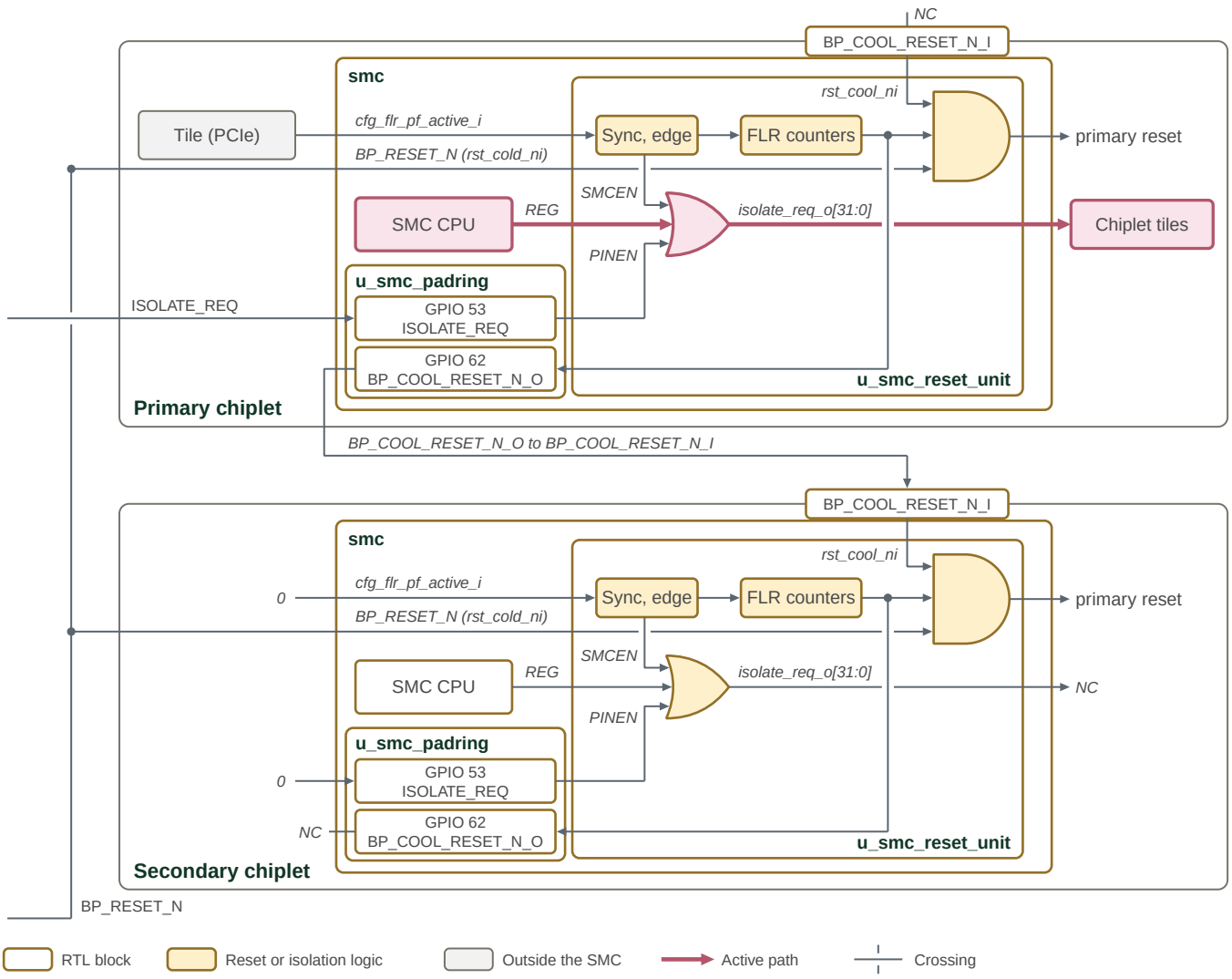


*Figure 54. SMC-driven FLR isolation*

Figure 55 shows isolation propagation for a tile request. A tile such as the PCIe controller raises `cfg_flr_pf_active_i`. Its synchronised rising edge sets `ISOLATE_REQ_SMC`, which isolates the subsystems enabled in `ISOLATE_REQ_SMCEN_REG`, and starts the FLR counters. When they expire, the FLR cool reset asserts the primary reset of the primary chiplet and, through `BP_COOL_RESET_N_O`, of the secondary chiplet.

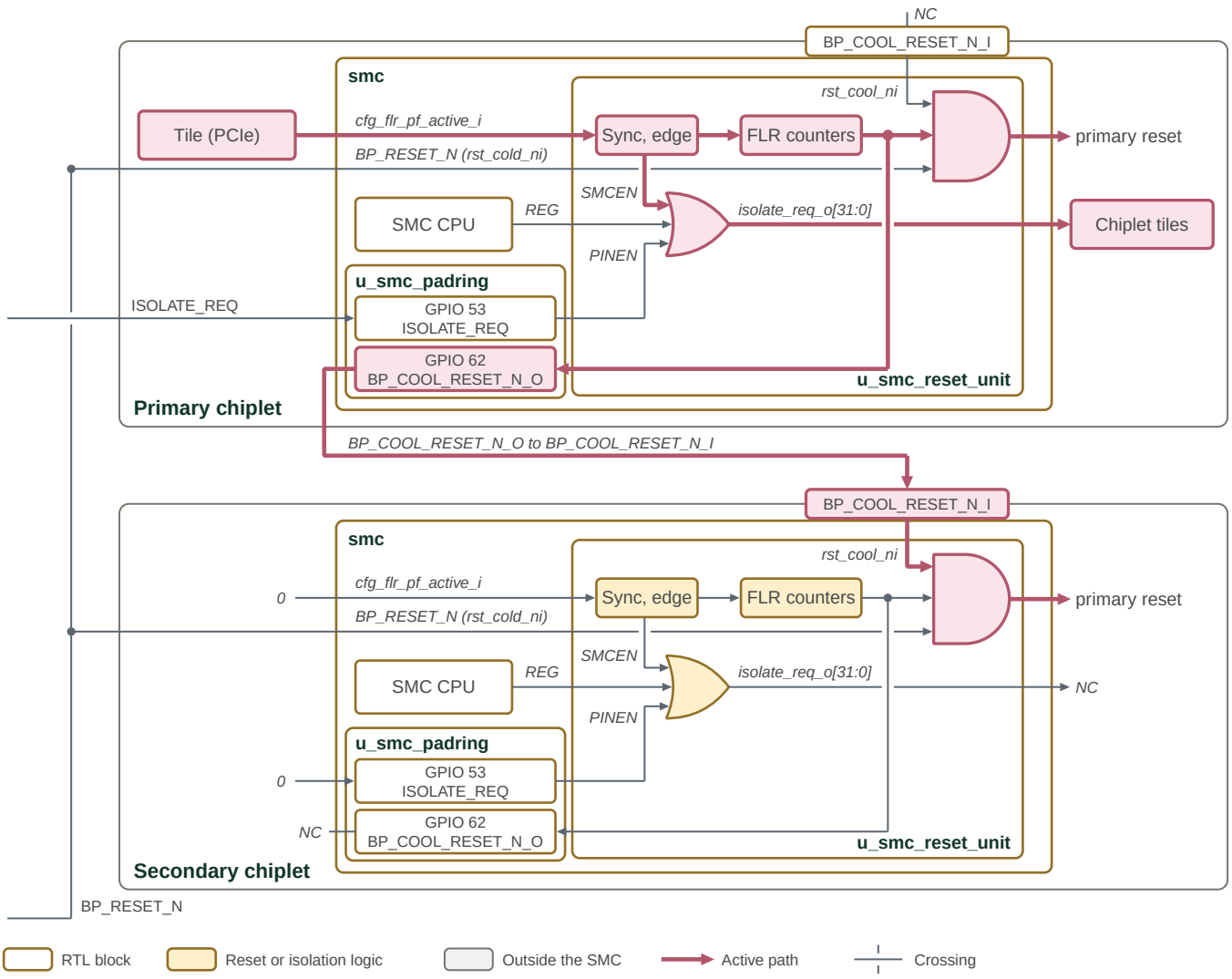


*Figure 55. Tile-request-driven FLR isolation*

Figure 56 shows isolation propagation for a GPIO request, in two numbered steps. First the board raises `ISOLATE_REQ`, which isolates the subsystems enabled in `ISOLATE_REQ_PINEN_REG`; then it asserts `BP_RESET_N`, which cold-resets both chiplets. `ISOLATE_REQ_PINEN_REG` is not reset by the cold reset while the pin is high, so the isolation holds across it.

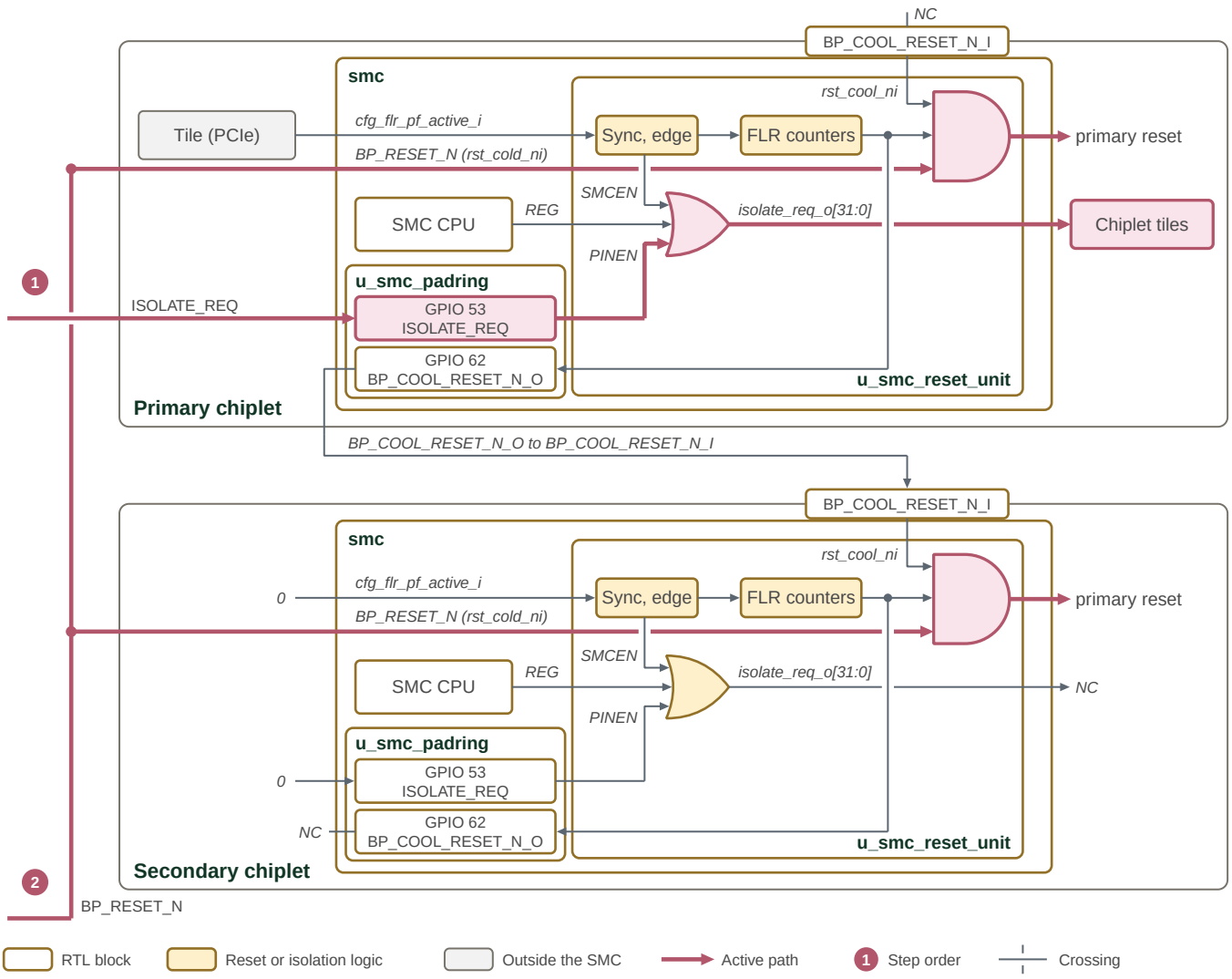


*Figure 56. GPIO-driven FLR isolation*

### Subsystem Controls and Clock Gating

The reset unit exports cold/warm reset, force-to-reference-clock and configuration, SRAM, debug and critical-signal hold controls for 32 subsystems. These are requests to the receiving subsystem, whose implementation defines the retained state. `SS_CONFIG_LOCK` and `SS_COLD_RESET_LOCK` mask writes to the corresponding configuration and cold-reset bits. `SS_RESET_COMPLETE` samples the return vector through a three-stage synchronizer; the unit does not sequence releases from it. The subsystem reset sequences are in the Programmer's Guide.

SMC blocks have their own clock-gating enables, activity detection and hysteresis controls. For example, DMA uses a six-bit hysteresis input and bus-aware wakeup. A gate suppresses clock transitions; it does not itself create a power island or establish that outstanding bus traffic is drained. Follow the owning block's quiescence requirements before reset, isolation or clock changes. The I2C, UART, I3C, AVSBus and telemetry windows answer `SLVERR` with read data `0xBADCAB1E` while their `CLOCK_GATE_CONTROL` enable is set, so software can detect an access to a gated peripheral rather than stall on it; the other gated blocks wake on bus activity and need no such response.

### Memory Map and Register Reference

### Memory Map

The following reset-unit map defines subsystem controls, isolation masks, visibility and FLR timing registers. Offsets are relative to the reset-unit base in the subsystem memory map. External state in the cool-reset wrapper follows the cold-reset behavior described above rather than the primary reset of the register-interface logic.

### Detailed Register Map

### Reset Unit Address Map

Subsystem reset control, hold and isolation controls, and reset-sequencing status registers.

*Table 853. Reset Unit Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x20 | SS_CONFIG | RW | - |
| 0x24 | SS_CONFIG_LOCK | RW | - |
| 0x40 | SS_COLD_RESET_N | RW | - |
| 0x44 | SS_WARM_RESET_N | RW | - |
| 0x48 | SS_CONFIG_HOLD | RW | - |
| 0x4C | SS_SRAM_HOLD | RW | - |
| 0x50 | SS_CRITICAL_HOLD | RW | - |
| 0x54 | SS_DEBUG_HOLD | RW | - |
| 0x60 | SS_RESET_COMPLETE | R | - |
| 0x70 | SS_COLD_RESET_LOCK | RW | - |
| 0x80 | SS_FORCE_TO_REF_CLK | RW | - |
| 0xA8 | SYNC_REG | RW | - |
| 0xB0 | ISOLATE_REQ_REG | RW | - |
| 0xB4 | ISOLATE_REQ_PINEN_REG | RW | - |
| 0xB8 | ISOLATE_REQ_SMC_REG | RW | - |
| 0xBC | ISOLATE_REQ_SMCEN_REG | RW | - |
| 0xC0 | ISOLATE_REQ_VIS | R | - |
| 0xC4 | ISOLATE_REQ_FLR_COUNTER_VALUE | RW | - |
| 0xC8 | ISOLATE_REQ_FLR_RESET_COUNTER_VALUE | RW | - |

### SS_CONFIG

*Table 854. reset_unit.SS_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `ss_config` | RW | 0x0 | Configuration registers to setup system configuration. Usage is integration-specific; see the adopter chiplet specification. |

### SS_CONFIG_LOCK

*Table 855. reset_unit.SS_CONFIG_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `config_lock` | RW1S | 0x0 | Written to lock down SS config. If bit 0 is written, then bit 0 of other SS config cannot be written to again |

### SS_COLD_RESET_N

*Table 856. reset_unit.SS_COLD_RESET_N fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `reset_n_n0_scan` | RW | 0x0 | Subsystem active low cold resets |

### SS_WARM_RESET_N

*Table 857. reset_unit.SS_WARM_RESET_N fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `reset_n_n0_scan` | RW | 0xFFFFFFFF | Subsystem active low warm resets |

### SS_CONFIG_HOLD

*Table 858. reset_unit.SS_CONFIG_HOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `configuration_state_hold` | RW | 0x0 | Used with warm_reset_n. Asserting ensures all registers configured during boot, including fuse values for export compliance, SRAM repair, other fuse values, etc. are preserved |

### SS_SRAM_HOLD

*Table 859. reset_unit.SS_SRAM_HOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `sram_hold` | RW | 0x0 | Used with warm_reset_n. Asserting ensures SRAMs, latch arrays and anything else that MBIST is used to test and initialize during cold reset are preserved |

### SS_CRITICAL_HOLD

*Table 860. reset_unit.SS_CRITICAL_HOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `critical_signal_hold` | RW | 0x0 | Used with warm_reset_n. Asserting ensures security, RAS, NMIs, perf counters, debug/trace and other subsystem specific registers are preserved |

### SS_DEBUG_HOLD

*Table 861. reset_unit.SS_DEBUG_HOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `debug_hold` | RW | 0x0 | Used with warm_reset_n. Debug hold signal to ensure any debug registers are preserved |

### SS_RESET_COMPLETE

*Table 862. reset_unit.SS_RESET_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `reset_complete` | R | 0x0 | Response signal from subsystem to indicate reset (or some other critical step) has completed. Each bit reads the synchronized live value of its subsystem's signal; it is not latched. |

### SS_COLD_RESET_LOCK

*Table 863. reset_unit.SS_COLD_RESET_LOCK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `cold_reset_lock` | RW1S | 0x0 | Written to lock down SS cold reset. If bit 0 is written, then bit 0 of other SS cold reset cannot be written to again |

### SS_FORCE_TO_REF_CLK

*Table 864. reset_unit.SS_FORCE_TO_REF_CLK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `force_ss_to_ref_clk_n` | RW | 0x0 | Forces clock to be ref clk. The ensures reset reaches all FFs within a subsystem in same clock cycle |

### SYNC_REG

*Table 865. reset_unit.SYNC_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `sync` | RW | 0x0 | Set to assert global sync bit |

### ISOLATE_REQ_REG

*Table 866. reset_unit.ISOLATE_REQ_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `isolate_req_reg` | RW | 0x0 | Isolate request. This prevents reset from being propagated to SS, keeping them alive in case of a cold reset |

### ISOLATE_REQ_PINEN_REG

*Table 867. reset_unit.ISOLATE_REQ_PINEN_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `isolate_req_pinen_reg` | RW | 0x0 | Isolate request pin enable. This enables the external isolate request pin to have an effect on ss isolate requests |

### ISOLATE_REQ_SMC_REG

*Table 868. reset_unit.ISOLATE_REQ_SMC_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `isolate_req_smc_reg` | RW | 0x0 | Isolate request smc. Asserted once cfg_flr_pf_active is asserted, de-asserted by sw |

#### ISOLATE_REQ_SMCEN_REG

*Table 869. reset_unit.ISOLATE_REQ_SMCEN_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `isolate_req_smcen_reg` | RW | 0x0 | Isolate request smc enable. This enables the FLR lock to have an effect on ss isolate requests |

#### ISOLATE_REQ_VIS

*Table 870. reset_unit.ISOLATE_REQ_VIS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `cool_reset_n_o` | R | 0x1 | Visibility bit for cool reset output |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `cool_reset_n_i` | R | 0x1 | Visibility bit for cool reset input |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `isolate_req_pin` | R | 0x1 | Visibility bit for isolate request pin |

#### ISOLATE_REQ_FLR_COUNTER_VALUE

*Table 871. reset_unit.ISOLATE_REQ_FLR_COUNTER_VALUE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `isolate_req_flr_counter_value` | RW | 0x0 | Number of cycles post cfg_flr_pf_active assertion is waited until cool_reset_n is asserted |

#### ISOLATE_REQ_FLR_RESET_COUNTER_VALUE

*Table 872. reset_unit.ISOLATE_REQ_FLR_RESET_COUNTER_VALUE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `isolate_req_flr_reset_counter_value` | RW | 0x0 | Number of cycles cool_reset_n is asserted for. NOTE: The cool reset flow requires this value to be greater than 0; a value of 0 will prevent initiation, regardless of ISOLATE_REQ_FLR_COUNTER_VALUE's value |

### 2.1.6.3. CPU

The SMC CPU subsystem contains four Rocket RV64GC cores, each with private instruction and data caches. The cores share a 1 MiB scratchpad and a 128 KiB boot ROM through the Chipyard-generated TileLink interconnect. AXI4 bridges connect the cluster to the SMC fabric.

This chapter describes the generated configuration, memory interfaces and CPU reset boundary. The Rocket Chip documentation provides background on the core and generator.

Figure 57 shows the CPU cluster, memory hierarchy, and fabric bridge.

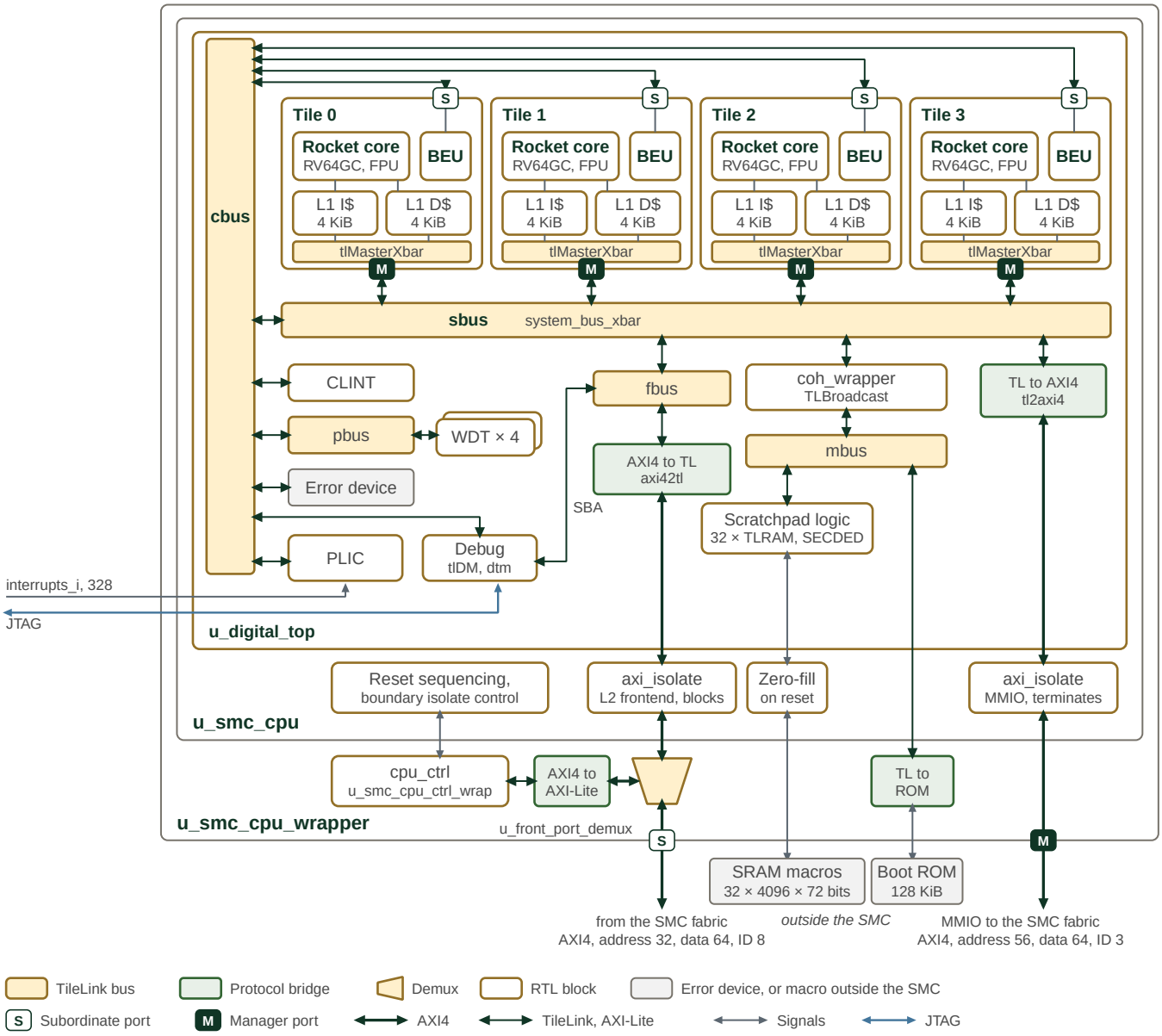


*Figure 57. SMC CPU cluster and TileLink-to-AXI integration*

The figure separates three implementation boundaries:

- `u_digital_top` contains the generated cluster and TileLink buses.
- `u_smc_cpu` adds AXI isolation, reset sequencing and scratchpad zero-fill.
- `u_smc_cpu_wrapper` adds the front-port demultiplexer, `cpu_ctrl` registers and boot ROM bridge.

Each tile connects to the system bus (`sbus`) as a manager and to the control bus (`cbus`) through its Bus Error Unit (BEU) subordinate port. Debug system bus accesses and the L2 frontend port enter `sbus` through the front bus (`fbus`). The coherence manager (`coh_wrapper`, a TileLink broadcast hub) connects to the memory bus (`mbus`), which serves the scratchpad and boot ROM. The PLIC receives four watchdog and four BEU interrupt sources; these paths are omitted from the figure.

Thick lines denote AXI4; thin dark lines denote TileLink, except for the AXI-Lite link to `cpu_ctrl`. The scratchpad contains 32 TLRAM banks with SECDED encode/decode logic. The SRAM macros, L1 tag/data arrays and boot ROM macro sit outside SMC. Memory interface ports carry their requests through the SMC boundary, including one 72-bit port for

each scratchpad bank.

### Processor Core

Table 873 summarizes the processor configuration.

*Table 873. SMC processor core features*

| Attribute | Description |
|---|---|
| Cores | 4, Rocket RV64GC |
| Base ISA | RV64GC (I, M, A, F, D, C extensions) |
| Additional configured extensions | Zba, Zbb, Zbs and conditional-zero instructions |
| Microarchitecture | In-order, 5-stage pipeline |
| Privilege Levels | M/S/U |
| Phys. Address | 56 bits |
| Reset Vector | Per-core, independently configurable (56b) |

### Local Memory

Table 874 lists the memory geometry in the generated cluster and the associated macro controls.

*Table 874. SMC cache and memory hierarchy*

| Component | Configured Size | Protection / role | Macro controls |
|---|---|---|---|
| L1 I-Cache per core | 4 KiB (32 sets × 2 ways) | Read-only instruction cache; tag/data parity | `icache_tag_cfg` / `icache_data_cfg` |
| L1 D-Cache per core | 4 KiB (32 sets × 2 ways) | Write-back data cache; tag/data SECDED ECC | `dcache_tag_cfg` / `dcache_data_cfg` |
| Local SRAM/Scratchpad | 1 MiB (32 banks) | Shared local memory; SECDED ECC | `scratch_ram_cfg` |
| Boot ROM | 128 KiB | Read-only, stores boot firmware | `rom_cfg` |

The sizes and associativity above are set by the Chipyard configuration and its generated memory interfaces. Changing them requires a matching generated cluster and memory implementation. The `*_cfg` signals configure the external memory macros; they are passed unchanged to each macro and do not resize caches or the scratchpad. Macro-specific repair and power controls depend on the selected memory implementation.

The cluster uses TileLink internally for the scratchpad and boot ROM, for the PLIC, CLINT and BEUs on the control bus, and for watchdog timers on the peripheral bus. At its boundary, the MMIO manager bridge converts TileLink to AXI4; the L2 frontend bridge converts incoming AXI4 to TileLink. The front-port branch serving `cpu_ctrl` uses AXI4-Lite. Both AXI4 ports connect to the SMC Fabric.

### Reset and Boot

`CPU_CTRL.RESET_CTRL` provides per-core reset controls and reset-pulse requests, plus separate uncore and debug-module reset controls. Each core has a 56-bit `RESET_VECTOR` register, reset to the boot ROM base `0xC004_0000`. The wrapper supplies these values to the generated cluster. While warm reset is asserted, it selects `0xC004_0000` instead of the programmed vectors.

Individual core reset clears the core's private L1 cache and TileLink client state without resetting the shared coherence manager. The current SMC Rocket integration provides no reset-aware coherence mechanism to drain outstanding transactions or preserve dirty cache data. The AXI boundary drain described below does not remove this limitation.

The scratchpad zero-fill sequencer starts when its memory-initialization reset releases. It writes zeros across the

scratchpad banks and holds the cores in reset until initialization completes. `smc_disable_sram_auto_init_i` bypasses this operation. `smc_init_mem_done_o` reports either completion or bypass; it does not certify memory repair, MBIST success or initialized contents when zero-fill is skipped. The integrator must provide suitable memory contents and ECC state before use when bypassing initialization.

### Memory Repair

Memory repair and memory built-in self-test (MBIST) are external DFT responsibilities. The external repair controller applies redundant memory elements as required by the selected SRAM implementation. The SMC fuse map provides no repair-data storage; the integrator supplies the repair data and its storage location.

The intended cold-boot sequence is fuse sensing, external repair, then MBIST before normal CPU use. SMC supplies the following sequencing and status interfaces; it does not implement the external repair algorithm.

**`smc_fuse_sense_done_o`**

Signals completion of fuse sensing into shadow registers. External DFT logic can use it to begin the repair/MBIST sequence.

**`ext_boot_seq_done_i`**

Reports completion of the external boot sequence. Its qualified value gates fuse-reset release together with fuse sensing; the repair and MBIST status bits themselves are not this handshake.

**`skip_mem_repair_o`**

Asserted by the synchronized isolation pin or the FLR-set `ISOLATE_REQ_SMC_REG` latch. Software writes can clear this latch but cannot set it. A request through the separate `ISOLATE_REQ_REG` register does not assert this output. External logic determines how it bypasses repair/MBIST.

**`BYPASS_SRAM_REPAIR` (`STRAPS_LO[13]`)**

Requests repair bypass. The ROM skips its repair-status check when this strap is set; the external integration owns the repair bypass behavior.

**`mem_repair_done` / `mem_repair_success`**

External completion and success status, visible in `DFX_CTRL_STATUS.STATUS_SMU` at `0xC000_B800`.

The ROM checks repair status during early startup, before normal stack use. It treats a clear `mem_repair_done` as repair not required rather than waiting for completion. When repair reports done, the ROM checks success; the `SMC_CONFIG` DFT-ignore-error setting can permit boot to continue after a reported failure. These firmware checks do not establish that repair has completed successfully. The external sequencer must enforce the platform's repair and MBIST requirements before reporting boot-sequence completion.

### System Interfaces

Table 875 identifies the protocols and roles of the CPU cluster interfaces.

*Table 875. SMC CPU system interfaces*

| Interface | Protocol | Description |
|---|---|---|
| MMIO | AXI4 | Manager port to the SMC fabric; 56-bit address, 64-bit data, 3-bit ID |
| L2 Frontend | AXI4 | Subordinate port into the cluster through the front bus; 32-bit SMC-local address, 64-bit data, 8-bit ID |
| Debug | JTAG | JTAG debug transport module into the debug module on the control bus; system bus access through the front bus |

### CPU AXI Isolation

The cluster is reset independently of the surrounding SMC fabric. If the cluster were reset while a transaction on one of its AXI ports was still in flight, the fabric on the other side would be left waiting on a response that never returns, hanging the bus, and could sample boundary signals while they glitch through reset. To avoid this, each AXI port is wrapped in an `axi_isolate` block that drains the outstanding transactions and holds the boundary at safe values before the reset is allowed to take effect.

Two instances wrap the AXI ports listed above, each sized for up to four outstanding transactions:

- the **L2 frontend** slave port, which blocks new transactions rather than terminating them, and
- the **MMIO** master port, which terminates new transactions presented while isolated with SLVERR responses to keep reset-time X values from propagating into the fabric.

The non-AXI status crossings (`wb_pc_valid`, `wb_reg_pc`, and the per-core `wdt_reset`) are clamped to constants over the same window so they cannot leak glitches either. The boundary also self-isolates on cold boot, releasing four cycles after SRAM initialization completes and the cores and uncore are out of reset.

The boundary is shared by all cores and stays isolated while any core or the uncore is in reset. Holding one core in reset therefore also terminates the other cores' MMIO accesses with SLVERR and stalls L2 frontend traffic until every core is released.

### Drain Handshake

The per-core reset sequence in the Programmer's Guide gives the firmware steps. A software reset through `CPU_CTRL.RESET_CTRL`, whether a core or uncore reset bit or a core reset pulse, proceeds as follows:

1. Firmware writes `RESET_CTRL`.
2. The control logic (`smc_cpu_ctrl_wrap`) raises `isolate_req_i` and withholds the reset.
3. Both `axi_isolate` instances drain their outstanding transactions and isolate.
4. The cluster raises `drained_o`, and the control logic applies the reset.
5. `isolate_req_i` stays high until every reset bit is released and every pulse completes. The boundary then reconnects.

### Timeout Recovery

`CPU_CTRL.RESET_TIMEOUT` limits how long a software-requested reset waits for the drain handshake. A `timeout_value` of zero disables the timer. When the timer expires, `timeout_mode=0` reports the timeout and continues to withhold the reset until the drain completes, then applies it. `timeout_mode=1` forces the reset and flushes both AXI isolation blocks.

`reset_timeout` remains high until the software reset request is removed, including after a recovery flush completes. `reset_applied` is high once the reset is applied by a completed drain or a force-mode timeout.

The flush clears outstanding-transaction records at the cluster boundary and discards late responses and write data until isolation is released. This lets the reset complete when the CPU cluster is unresponsive but the surrounding fabric remains operational.

| | |
|---|---|
| **WARNING** | The flush cannot cancel an address request or write-data beat while `VALID` is high and `READY` is low. It waits for that transfer before flushing the channel. If `READY` never goes high, isolation cannot complete.<br><br>The flush clears only the CPU isolation blocks. It does not clear state elsewhere in the SMC fabric. |

> The cluster reset may complete while another block remains unresponsive or holds a response, leaving that route or AXI ID blocked, so that block must be recovered separately or through the eventual cold reset.

#### Reset-Domain Crossing

The flops driving the isolate-control logic are held by the cluster's uncore/core reset, while the flops inside `axi_isolate` are held by the Primary Reset (see Clock and Reset Management ). The uncore/core reset is driven from two sources:

- a Warm Reset from a watchdog timeout, which is **asynchronous** to the isolate logic, and
- a CPU CSR write (the reset-control register in `smc_cpu_ctrl_wrap`), which is **synchronous**.

Because the two sets of flops sit in different reset domains, the asynchronous warm-reset path is a reset-domain crossing into `axi_isolate`. This is acceptable: only the watchdog asserts the warm reset asynchronously, and a watchdog timeout is followed by a cold reset, so the brief crossing window has no functional consequence. The CSR path is synchronous and crosses no domains.

#### Interrupt Integration

All interrupt wiring—MMIO bases for PLIC, CLINT, watchdog, and bus-error units; the authoritative `cpu_interrupts_o` map; PLIC/CLINT register documentation; and BEU direct interrupt delivery—is documented in Interrupts.

#### Watchdog Interface

Each core has a watchdog. See Watchdog Timers for operation and the register reference.

#### Bus Error Reporting

Each core has a Bus Error Unit (BEU) at `0xC801_0000 + N * 0x1000`, for N = 0 through 3. It reports errors seen by that core's L1 caches:

- instruction-cache correctable errors,
- instruction-cache bus errors,
- data-cache correctable errors,
- data-cache uncorrectable errors, and
- data-cache bus errors.

An error response from the SMC fabric reaches the BEU only as a cache bus error; the BEU does not distinguish decode, slave or timeout errors. It is not a general AXI transaction monitor and has no independent AXI timeout detector or transaction-data log.

`ENABLE` selects events that capture `CAUSE` and `PHYS_ADDR` while `CAUSE` is zero. Instruction-cache correctable errors capture a physical address of zero. The register named `ACCRUED_ENABLE` is instead sticky event status: events accumulate independently of `ENABLE`, and ordinary software writes replace the bits. `PLIC_ENABLE` and `LOCAL_ENABLE` select from that accrued status for the PLIC and direct-core interrupt paths. Capture the diagnostic state before clearing status or rearming cause capture. The generated register map below lists the implemented fields. Delivery and clearing are described in Bus Error Units.

### Debug

The generated memory map below describes the hart-facing Debug Module memory region. Only the harts of the CPU cluster reach it, over the cluster's own control bus. The SMC fabric does not route it, so system, SEP and other SMC initiators receive DECERR at every address in the region. Debugger-facing registers such as `dmcontrol`, `dmstatus`, `abstractcs` and `sbcs` are accessed through the Debug Module Interface (DMI), not through that memory region. Debug Module Configuration describes them.

JTAG reaches the cluster's debug transport module and debug module. The configured debug facilities provide per-core register access, program-buffer execution, breakpoints and system bus access. Debug system bus transactions enter through `fbus` and use the cluster's interconnect and reset/isolation conditions.

Debug-module reset is separately controlled by `CPU_CTRL.RESET_CTRL`; its reset value holds the debug module in reset. Separate control does not make debug memory accesses independent of cluster reset or boundary isolation. Per-core boot addresses are controlled by the `RESET_VECTOR` registers described above.

### Debug Module Configuration

The Debug Module implements version 0.13 of the RISC-V External Debug Support specification for the four harts of the CPU cluster. A debugger discovers its configuration by reading DMI registers. Table 876 lists the fixed capability values those registers return, and Table 877 lists which DMI registers are implemented.

*Table 876. SMC Debug Module capability readbacks*

| Register | Field | Value | Meaning |
|---|---|---|---|
| `dmstatus` | `version` | 2 | Debug Specification 0.13 |
| `dmstatus` | `authenticated` | 1 | No authentication is required |
| `dmstatus` | `hasresethaltreq` | 1 | Halt-on-reset is supported |
| `dmstatus` | `confstrptrvalid` | 0 | No configuration string |
| `dmstatus` | `impebreak` | 1 | An implicit `ebreak` follows the program buffer |
| `dmcontrol` | `hartsello` | 2 bits | Harts 0 to 3; `hartselhi` reads 0 |
| `abstractcs` | `progbufsize` | 16 | `progbuf0` to `progbuf15` |
| `abstractcs` | `datacount` | 4 | `data0` to `data3` |
| `abstractauto` | `autoexecdata` | 4 bits | Auto-execution from `data0` to `data3` |
| `abstractauto` | `autoexecprogbuf` | 16 bits | Auto-execution from `progbuf0` to `progbuf15` |
| `sbcs` | `sbversion` | 1 | System bus access as defined by Debug Specification 0.13 |
| `sbcs` | `sbasize` | 56 | System bus address width in bits |
| `sbcs` | `sbaccess8`, `sbaccess16`, `sbaccess32`, `sbaccess64` | 1 | 8-, 16-, 32- and 64-bit system bus accesses |
| `sbcs` | `sbaccess128` | 0 | No 128-bit system bus access |

The `command` register accepts only the Access Register command (`cmdtype` 0). With `transfer` set, Access Register reaches the general-purpose registers `x0` to `x31` (`regno` 0x1000 to 0x101F) with an `aarsize` of 2 or 3 (32 or 64 bits). With `transfer` clear, any `regno` and size is accepted. Any other command type, and any transfer outside those registers and sizes, sets `abstractcs.cmderr` to 2 (not supported). The debugger reaches CSRs and floating-point registers by executing instructions from the program buffer. Reading `command` returns the last value written.

`hawindowsel` reads 0 and ignores writes, so `hawindow` bits 3:0 hold the hart array mask for harts 0 to 3. `setresethaltreq` and `clrresethaltreq` in `dmcontrol` set and clear halt-on-reset for the selected harts.

DMI address 0x32 provides the halt-group fields of the `dmcs2` register, which later versions of the specification define. A

write with `hgwrite` set, `hgselect` clear and `haltgroup` 0 or 1 removes the selected harts from halt group 1 or adds them to it. When one hart in halt group 1 halts, the Debug Module halts the other harts in that group. A read returns the group of the hart selected by `hartsel` in `haltgroup`. External trigger groups (`hgselect` set) are not supported.

DMI registers that the Debug Module does not implement read as 0 and ignore writes.

*Table 877. SMC Debug Module DMI register inventory*

| DMI address | Register | Implementation |
|---|---|---|
| 0x04-0x07 | `data0-data3` | Implemented |
| 0x08-0x0F | `data4-data11` | Not implemented |
| 0x10 | `dmcontrol` | Implemented; `hartsello` bits 17:16 only, `hartselhi` reads 0 |
| 0x11 | `dmstatus` | Implemented |
| 0x12 | `hartinfo` | Not implemented |
| 0x13 | `haltsum1` | Implemented; bit 0 only |
| 0x14 | `hawindowsel` | Reads 0 |
| 0x15 | `hawindow` | Implemented; bits 3:0 only |
| 0x16 | `abstractcs` | Implemented |
| 0x17 | `command` | Implemented; Access Register only |
| 0x18 | `abstractauto` | Implemented; `autoexecdata` bits 3:0 only |
| 0x19-0x1C | `confstrptr0-confstrptr3` | Not implemented |
| 0x1D | `nextdm` | Not implemented; this is the only Debug Module |
| 0x20-0x2F | `progbuf0-progbuf15` | Implemented |
| 0x30 | `authdata` | Not implemented |
| 0x32 | `dmcs2` | Implemented; halt groups 0 and 1 only |
| 0x34 | `haltsum2` | Not implemented |
| 0x35 | `haltsum3` | Not implemented |
| 0x37 | `sbaddress3` | Not implemented |
| 0x38 | `sbcs` | Implemented |
| 0x39-0x3A | `sbaddress0-sbaddress1` | Implemented |
| 0x3B | `sbaddress2` | Not implemented |
| 0x3C-0x3D | `sbdata0-sbdata1` | Implemented |
| 0x3E-0x3F | `sbdata2-sbdata3` | Not implemented |
| 0x40 | `haltsum0` | Implemented; bits 3:0 only |

### Abstract Command Writes While cmderr Is Set

The Debug Module departs from the RISC-V External Debug Support specification (version 0.13) for writes to `command` while `abstractcs.cmderr` is nonzero. The specification ignores those writes. In this Debug Module, a `command` write made while no abstract command is executing (`abstractcs.busy` is 0) has two effects:

- the written value replaces the stored command, whatever `cmderr` holds, and
- a command type other than Access Register (`cmdtype` other than 0) sets `cmderr` to 2 (not supported), replacing the error code it held.

No abstract command executes while `cmderr` is nonzero. Once `cmderr` is cleared, an access that `abstractauto` selects re-executes the stored command, which is the last value written to `command`.

A `command` write therefore discards the error code of a failed command unless `abstractcs` was read first. The Programmer's Guide gives the debugger sequence for clearing `cmderr`.

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### CPU Control Address Map

Per-core reset control and status, reset timing, chip configuration, mailbox, and boot-ROM stub registers for the SMC CPU cluster.

**NOTE**

**Register Arrays:**

- **RESET_VECTOR[4]**: 4 registers, base 0x0, stride 0x8
- **SCRATCH[16]**: 16 registers, base 0x80, stride 0x8
- **WB_PC_CORE0[8]**: 8 registers, base 0x100, stride 0x8
- **WB_PC_CORE1[8]**: 8 registers, base 0x140, stride 0x8
- **WB_PC_CORE2[8]**: 8 registers, base 0x180, stride 0x8
- **WB_PC_CORE3[8]**: 8 registers, base 0x1C0, stride 0x8
- **MUTEX[4]**: 4 registers, base 0x240, stride 0x8
- **SEMA[4]**: 4 registers, base 0x260, stride 0x8
- **DUMMY_ROM_NULL[4]**: 4 registers, base 0x2A0, stride 0x8

*Table 878. CPU Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x18 | RESET_VECTOR[4] | RW | - |
| 0x20 | RESET_CTRL | RW | - |
| 0x28 | CORE_RESET_PULSE_COUNT | RW | - |
| 0x30 | RESET_TIMEOUT | RW | - |
| 0x40 | REFERENCE_COUNTER | RW | - |
| 0x50 | WDT_TIMEOUT | RW | - |
| 0x58 | WDT_TIMEOUT_RESET | RW | - |
| 0x60 | TEST_CTRL | R | - |
| 0x80 - 0xF8 | SCRATCH[16] | RW | - |
| 0x100 - 0x138 | WB_PC_CORE0[8] | R | - |
| 0x140 - 0x178 | WB_PC_CORE1[8] | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x180 - 0x1B8 | WB_PC_CORE2[8] | R | - |
| 0x1C0 - 0x1F8 | WB_PC_CORE3[8] | R | - |
| 0x200 | SMC_ATTRIBUTES | R | - |
| 0x240 - 0x258 | MUTEX[4] | RW | - |
| 0x260 - 0x278 | SEMA[4] | RW | - |
| 0x280 | DUMMY_ROM_0 | RW | - |
| 0x288 | DUMMY_ROM_1 | RW | - |
| 0x290 | DUMMY_ROM_2 | RW | - |
| 0x298 | DUMMY_ROM_3 | RW | - |
| 0x2A0 - 0x2B8 | DUMMY_ROM_NULL[4] | RW | - |

### RESET_VECTOR[4]

*Table 879. cpu_ctrl.RESET_VECTOR[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `vector` | RW | 0xC0040000 | - |

### RESET_CTRL

*Table 880. cpu_ctrl.RESET_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:25 | `Reserved` | - | - | Reserved |
| 24 | `debug_reset_n_n0_scan` | RW | 0x0 | Debug Module Reset, active low |
| 23:9 | `Reserved` | - | - | Reserved |
| 8 | `uncore_reset_n_n0_scan` | RW | 0x1 | Cluster Uncore Reset, active low. Writing this register will only reset the uncore portion in the chipyard cluster. WARNING: using this can cause any outstanding axi requests to and from the cluster to cause timeouts/exceptions. |
| 7 | `core3_reset_pulse_start_n0_scan` | RW | 0x0 | Writing a 1 to this register triggers a core reset pulse on core 3, timed by CORE_RESET_PULSE_COUNT. A 0 in core3_reset_n_n0_scan holds the core in reset for the whole pulse. |
| 6 | `core2_reset_pulse_start_n0_scan` | RW | 0x0 | Writing a 1 to this register triggers a core reset pulse on core 2, timed by CORE_RESET_PULSE_COUNT. A 0 in core2_reset_n_n0_scan holds the core in reset for the whole pulse. |
| 5 | `core1_reset_pulse_start_n0_scan` | RW | 0x0 | Writing a 1 to this register triggers a core reset pulse on core 1, timed by CORE_RESET_PULSE_COUNT. A 0 in core1_reset_n_n0_scan holds the core in reset for the whole pulse. |
| 4 | `core0_reset_pulse_start_n0_scan` | RW | 0x0 | Writing a 1 to this register triggers a core reset pulse on core 0, timed by CORE_RESET_PULSE_COUNT. A 0 in core0_reset_n_n0_scan holds the core in reset for the whole pulse. |
| 3 | `core3_reset_n_n0_scan` | RW | 0x1 | Reset control for the CPU, active low. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | `core2_reset_n_n0_scan` | RW | 0x1 | Reset control for the CPU, active low. |
| 1 | `core1_reset_n_n0_scan` | RW | 0x1 | Reset control for the CPU, active low. |
| 0 | `core0_reset_n_n0_scan` | RW | 0x1 | Reset control for the CPU, active low. |

### CORE_RESET_PULSE_COUNT

*Table 881. cpu_ctrl.CORE_RESET_PULSE_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:36 | `Reserved` | - | - | Reserved |
| 35:32 | `core_resets_done` | R | 0xF | Logic low while performing core reset, logic high otherwise. |
| 31:16 | `post_reset_count` | RW | 0x10 | Cycles (clk_smc), minus one, that a core reset pulse holds the core in reset. |
| 15:0 | `pre_reset_count` | RW | 0x8 | Cycles (clk_smc), minus one, from the start of a core reset pulse to reset assertion. |

### RESET_TIMEOUT

*Table 882. cpu_ctrl.RESET_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:37 | `Reserved` | - | - | Reserved |
| 36 | `reset_timeout` | R | 0x0 | Status for the active software reset request: high after the request exceeds timeout_value and remains high until the request is deasserted. Set in both modes. |
| 35:33 | `Reserved` | - | - | Reserved |
| 32 | `reset_applied` | R | 0x0 | Live status: high while a requested software reset is applied at the cluster, either after drain completes or after a force-mode timeout. |
| 31:17 | `Reserved` | - | - | Reserved |
| 16 | `timeout_mode` | RW | 0x0 | Timeout mode: 0 = report the timeout and keep withholding the reset until the drain completes, then apply it; 1 = force the reset on timeout. |
| 15:0 | `timeout_value` | RW | 0x0 | Cycles (clk_smc) a software reset may stay pending (withheld, waiting for drain) before timeout. 0 = disabled, wait indefinitely. |

### REFERENCE_COUNTER

*Table 883. cpu_ctrl.REFERENCE_COUNTER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rc` | RW | 0x0 | Reference counter for the CPU. Write once, then poll this register until the readback reflects the written value before writing again. The counter keeps advancing, so the readback is at or just above the written value rather than equal to it. |

### WDT_TIMEOUT

*Table 884. cpu_ctrl.WDT_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x4000 | Cycle count value that 2nd stage watchdog timer timeout should occur at. Defaults to 16384 cycles |

### WDT_TIMEOUT_RESET

*Table 885. cpu_ctrl.WDT_TIMEOUT_RESET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `reset_cycle_count_3` | RW 1P | 0x0 | Reset current cycle count value in 2nd stage watchdog timer |
| 2 | `reset_cycle_count_2` | RW 1P | 0x0 | Reset current cycle count value in 2nd stage watchdog timer |
| 1 | `reset_cycle_count_1` | RW 1P | 0x0 | Reset current cycle count value in 2nd stage watchdog timer |
| 0 | `reset_cycle_count_0` | RW 1P | 0x0 | Reset current cycle count value in 2nd stage watchdog timer |

### TEST_CTRL

*Table 886. cpu_ctrl.TEST_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `data` | R | 0x0 | Test control register for the CPU |

### SCRATCH[16]

*Table 887. cpu_ctrl.SCRATCH[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `data` | RW | 0x0 | Scratch register for the CPU |

### WB_PC_CORE0[8]

*Table 888. cpu_ctrl.WB_PC_CORE0[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Reserved` | - | - | Reserved |
| 57:0 | `pc` | R | 0x0 | Writeback PC value |

### WB_PC_CORE1[8]

*Table 889. cpu_ctrl.WB_PC_CORE1[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Reserved` | - | - | Reserved |
| 57:0 | `pc` | R | 0x0 | Writeback PC value |

### WB_PC_CORE2[8]

*Table 890. cpu_ctrl.WB_PC_CORE2[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Reserved` | - | - | Reserved |
| 57:0 | `pc` | R | 0x0 | Writeback PC value |

### WB_PC_CORE3[8]

*Table 891. cpu_ctrl.WB_PC_CORE3[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Reserved` | - | - | Reserved |
| 57:0 | `pc` | R | 0x0 | Writeback PC value |

### SMC_ATTRIBUTES

*Table 892. cpu_ctrl.SMC_ATTRIBUTES fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:46 | `Reserved` | - | - | Reserved |
| 45:40 | `num_mailboxes` | R | - | - |
| 39:36 | `mailbox_depth` | R | - | - |
| 35:33 | `num_cores` | R | - | - |
| 32 | `no_output_remap` | R | - | - |
| 31:26 | `sram_size` | R | - | Total SRAM size is 2 ^ value bytes |
| 25:17 | `num_cpu_interrupts` | R | - | - |
| 16:8 | `num_ext_interrupts` | R | - | - |
| 7:1 | `Reserved` | - | - | Reserved |
| 0 | `chiplet_is_primary` | R | - | Indicates if this chiplet is primary (1) or secondary (0). Driven by strap. |

### MUTEX[4]

*Table 893. cpu_ctrl.MUTEX[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Reserved` | - | - | Reserved |
| 0 | `mutex` | RW | 0x1 | HW mutex. Reads will attempt to acquire mutex, 1 on success. If the mutex is already acquired, the read will return 0. To release the mutex, write any value to the register. |

### SEMA[4]

*Table 894. cpu_ctrl.SEMA[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:16 | `Reserved` | - | - | Reserved |
| 15:0 | `sema` | RW | 0x0 | 16-bit semaphore value to inc/dec. Writing to this register will inc/dec the semaphore value. The written value is treated as a signed number using 2s compliment. |

### DUMMY_ROM_0

*Table 895. cpu_ctrl.DUMMY_ROM_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `dummy_rom_word_0` | RW | 0x1450513 00000517 | Word 0 of WFI loop |

#### DUMMY_ROM_1

*Table 896. cpu_ctrl.DUMMY_ROM_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `dummy_rom_word_1` | RW | 0xFFF0069330551073 | Word 1 of WFI loop |

#### DUMMY_ROM_2

*Table 897. cpu_ctrl.DUMMY_ROM_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `dummy_rom_word_2` | RW | 0x105000733046B073 | Word 2 of WFI loop |

#### DUMMY_ROM_3

*Table 898. cpu_ctrl.DUMMY_ROM_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `dummy_rom_word_3` | RW | 0xFFDFF06F | Word 3 of WFI loop |

#### DUMMY_ROM_NULL[4]

*Table 899. cpu_ctrl.DUMMY_ROM_NULL[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `dummy_rom_null` | RW | 0x0 | Cache line size is 64 bytes so want to ensure no random data is read when reading dummy rom |

### CLINT Address Map

Core-Local Interruptor: software-interrupt, timer-compare, and machine-time registers.

**NOTE**

**Register Arrays:**

- **MSIP[4]**: 4 registers, base 0x0, stride 0x4
- **MTIMECMP[4]**: 4 registers, base 0x4000, stride 0x8

*Table 900. CLINT Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xC | MSIP[4] | RW | - |
| 0x4000 - 0x4018 | MTIMECMP[4] | RW | - |
| 0xBFF8 | MTIME | RW | - |

#### MSIP[4]

*Table 901. clint.MSIP[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `rsvd0` | R | 0x0 | Reserved |
| 0 | `value` | RW | 0x0 | MSIP bit for core |

#### MTIMECMP[4]

*Table 902. clint.MTIMECMP[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `count` | RW | - | Timer compare value. The core's machine timer interrupt is asserted while MTIME is greater than or equal to this value, and deasserts when software sets this value above MTIME. |

#### MTIME

*Table 903. clint.MTIME fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `count` | RW | - | Cycle count (time) value |

### Debug Module Hart Interface

Memory region through which a hart in Debug Mode exchanges state with the Debug Module: handshake mailboxes, generated instructions, program buffer, data registers, per-hart flags and the debug ROM. Only the harts of the CPU cluster reach this region; SMC bus initiators outside the cluster do not. The debugger-facing Debug Module registers (dmcontrol, dmstatus, abstractcs, sbcs, …) are reached over the Debug Module Interface and do not appear in this map.

**NOTE**

**Register Arrays:**

- **ABSTRACT[5]**: 5 registers, base 0x328, stride 0x4
- **PROGBUF[16]**: 16 registers, base 0x33C, stride 0x4
- **DATA[4]**: 4 registers, base 0x380, stride 0x4
- **ROM[26]**: 26 registers, base 0x800, stride 0x4

*Table 904. Debug Module Hart Interface register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x100 | HALTED | W | - |
| 0x104 | GOING | W | - |
| 0x108 | RESUMING | W | - |
| 0x10C | EXCEPTION | W | - |
| 0x300 | WHERETO | R | - |
| 0x328 - 0x338 | ABSTRACT[5] | R | - |
| 0x33C - 0x378 | PROGBUF[16] | RW | - |
| 0x37C | IMPEBREAK | R | - |
| 0x380 - 0x38C | DATA[4] | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x400 | FLAGS | R | One flag byte per hart, at byte offset equal to the hart ID. Bit 0 is go (execute the command at WHERETO); bit 1 is resume (leave Debug Mode). Bits 7:2 read as 0. |
| 0x800 - 0x864 | ROM[26] | R | - |

### HALTED

*Table 905. debug_module.HALTED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `hartid` | W | - | Debug ROM causes the hart to write its hart ID here when it is in Debug Mode. |

### GOING

*Table 906. debug_module.GOING fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `hartid` | W | - | Debug ROM causes the hart to write 0 here when it begins executing Debug Mode instructions. |

### RESUMING

*Table 907. debug_module.RESUMING fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `hartid` | W | - | Debug ROM causes the hart to write its hart ID here when it leaves Debug Mode. |

### EXCEPTION

*Table 908. debug_module.EXCEPTION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `hartid` | W | - | Debug ROM causes the hart to write 0 here if it takes an exception in Debug Mode. |

### WHERETO

*Table 909. debug_module.WHERETO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `instr` | R | - | Jump instruction filled in by the Debug Module to direct the hart in Debug Mode. |

#### ABSTRACT[5]

*Table 910. debug_module.ABSTRACT[5] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `instr` | R | - | Instruction generated by the Debug Module for the current abstract command. |

#### PROGBUF[16]

*Table 911. debug_module.PROGBUF[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x0 | Program buffer word shared with the debugger. Byte accesses are supported. |

#### IMPEBREAK

*Table 912. debug_module.IMPEBREAK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `instr` | R | 0x100073 | Implicit EBREAK instruction following the program buffer. |

#### DATA[4]

*Table 913. debug_module.DATA[4] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x0 | Abstract command data word shared with the debugger. Byte accesses are supported. |

#### FLAGS

*Table 914. debug_module.FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `flags_3` | R | - | Hart 3 go/resume flags. |
| 23:16 | `flags_2` | R | - | Hart 2 go/resume flags. |
| 15:8 | `flags_1` | R | - | Hart 1 go/resume flags. |
| 7:0 | `flags_0` | R | - | Hart 0 go/resume flags. |

#### ROM[26]

*Table 915. debug_module.ROM[26] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | Debug ROM word. Holds the fixed Debug Mode entry, exception and resume handlers. |

### Bus Error Unit Address Map

Status registers reporting correctable and uncorrectable I-cache, D-cache, and TileLink bus errors.

*Table 916. Bus Error Unit Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CAUSE | RW | - |
| 0x8 | PHYS_ADDR | RW | - |
| 0x10 | ENABLE | RW | - |
| 0x18 | PLIC_ENABLE | RW | - |
| 0x20 | ACCRUED_ENABLE | RW | Sticky accrued error status. The legacy name ACCRUED_ENABLE does not denote an enable mask. Error events set these bits independently of ENABLE. Ordinary software writes replace the bits (write zero to clear) and take priority over simultaneous events. PLIC_ENABLE and LOCAL_ENABLE mask this status for interrupt delivery. |
| 0x28 | LOCAL_ENABLE | RW | - |

### CAUSE

*Table 917. bus_error_unit.CAUSE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2:0 | `data` | RW | 0x0 | Cause register. Enumerates different kinds of errors |

### PHYS_ADDR

*Table 918. bus_error_unit.PHYS_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `value` | RW | - | Physical address captured with CAUSE when an enabled error arrives and CAUSE is zero. Software may overwrite the value; writes take priority over capture. No reset value |

### ENABLE

*Table 919. bus_error_unit.ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:8 | `Reserved` | - | - | Reserved |
| 7 | `dcache_uncorrectable` | RW | 0x1 | Enable recording error information for DCache ECC uncorrectable bus error. Enabled on reset. |
| 6 | `dcache_correctable` | RW | 0x1 | Enable recording error information for DCache ECC correctable bus error. Enabled on reset. |
| 5 | `dcache_tlbus` | RW | 0x1 | Enable recording error information for DCache Tilelink bus error. Enabled on reset. |
| 4 | `rsvd4` | RW | 0x0 | Reserved |
| 3 | `rsvd3` | RW | 0x0 | Reserved |
| 2 | `icache_correctable` | RW | 0x1 | Enable recording error information for ICache ECC correctable bus error. Enabled on reset. |
| 1 | `icache_tlbus` | RW | 0x1 | Enable recording error information for ICache Tilelink bus error. Enabled on reset. |
| 0 | `rsvd0` | RW | 0x0 | Reserved |

### PLIC_ENABLE

*Table 920. bus_error_unit.PLIC_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:8 | `Reserved` | - | - | Reserved |
| 7 | `dcache_uncorrectable` | RW | 0x0 | PLIC interrupt enable mask for DCache ECC uncorrectable bus error. |
| 6 | `dcache_correctable` | RW | 0x0 | PLIC interrupt enable mask for DCache ECC correctable bus error. |
| 5 | `dcache_tlbus` | RW | 0x0 | PLIC interrupt enable mask for DCache Tilelink bus error. |
| 4 | `rsvd4` | RW | 0x0 | Reserved |
| 3 | `rsvd3` | RW | 0x0 | Reserved |
| 2 | `icache_correctable` | RW | 0x0 | PLIC interrupt enable mask for ICache ECC correctable bus error. |
| 1 | `icache_tlbus` | RW | 0x0 | PLIC interrupt enable mask for ICache Tilelink bus error. |
| 0 | `rsvd0` | RW | 0x0 | Reserved |

### ACCRUED_ENABLE

*Table 921. bus_error_unit.ACCRUED_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:8 | `Reserved` | - | - | Reserved |
| 7 | `dcache_uncorrectable` | RW | 0x0 | Sticky accrued status for DCache ECC uncorrectable bus error. |
| 6 | `dcache_correctable` | RW | 0x0 | Sticky accrued status for DCache ECC correctable bus error. |
| 5 | `dcache_tlbus` | RW | 0x0 | Sticky accrued status for DCache Tilelink bus error. |
| 4 | `rsvd4` | RW | 0x0 | Reserved |
| 3 | `rsvd3` | RW | 0x0 | Reserved |
| 2 | `icache_correctable` | RW | 0x0 | Sticky accrued status for ICache ECC correctable bus error. |
| 1 | `icache_tlbus` | RW | 0x0 | Sticky accrued status for ICache Tilelink bus error. |
| 0 | `rsvd0` | RW | 0x0 | Reserved |

### LOCAL_ENABLE

*Table 922. bus_error_unit.LOCAL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:8 | `Reserved` | - | - | Reserved |
| 7 | `dcache_uncorrectable` | RW | 0x0 | Core local interrupt enable mask for DCache ECC uncorrectable bus error. |
| 6 | `dcache_correctable` | RW | 0x0 | Core local interrupt enable mask for DCache ECC correctable bus error. |
| 5 | `dcache_tlbus` | RW | 0x0 | Core local interrupt enable mask for DCache Tilelink bus error. |
| 4 | `rsvd4` | RW | 0x0 | Reserved |
| 3 | `rsvd3` | RW | 0x0 | Reserved |
| 2 | `icache_correctable` | RW | 0x0 | Core local interrupt enable mask for ICache ECC correctable bus error. |
| 1 | `icache_tlbus` | RW | 0x0 | Core local interrupt enable mask for ICache Tilelink bus error. |
| 0 | `rsvd0` | RW | 0x0 | Reserved |

### 2.1.6.4. Boot ROM

The SMC maps a 128 KiB read-only region at `0xC004_0000` through `0xC005_FFFF`. The cold-reset CPU vector is `0xC004_0000`.

#### ROM Architecture

In the current four-core implementation, the CPU reaches the ROM over its internal TileLink memory path. `smc_cpu_wrapper` converts that path to the macro-facing `smc_rom_intf_req_o` and `smc_rom_intf_rsp_i` interface. Requests carry a clock, enable, and 14-bit 64-bit-word address; the write controls are tied to read mode. Responses return one 64-bit word. The ROM macro boundary is not an AXI-Lite interface and is separate from the CPU MMIO AXI and SMC peripheral AXI-Lite paths.

#### Hardware Configuration and Integrity

`OCAH4CORECluster_mems` passes the 11-bit `rom_cfg_i` value unchanged to the ROM wrapper's `mem_cfg_i`, where it is cast to `prim_rom_pkg::rom_cfg_t` and forwarded to `prim_rom`. The OCAH logic assigns no repair or technology-specific meaning to those bits. The reference `smc_ip_integration.sv` ties the value to zero.

On the response path, asserted `rom_flip_endianness_i` reverses the order of the eight bytes in each 64-bit ROM word; deasserted passes the word through unchanged. In `smc.sv`, this control comes from the ROM endianness bit in the eFuse shadow-register output.

#### Boot Flow

The production ROM initializes SRAM and the C runtime, then services OCCP requests over target-mode I2C and I3C. The SMC Boot ROM appendix of the Programmer's Guide describes the startup checks, lifecycle policy and boot-handoff commands.

The production ROM image fills the 128 KiB hardware region; its writable data runs from SRAM at `0xC006_0000`. The SMC CPU bring-up section of the Programmer's Guide gives the linker layout and the initialization it requires.

#### Firmware Reference

The production ROM firmware is documented in the SMC Boot ROM appendix of the Programmer's Guide.

### 2.1.6.5. Interconnect

The SMC fabric provides interconnect infrastructure with address management, traffic filtering, and security enforcement mechanisms.

#### Topology

#### Fabric Topology

The input fabric merges the SMC's own initiators, filters the system AXI input and splits traffic between the SMC window and the rest of the system. The local fabric connects the traffic for the SMC window to the CPU front port and the SMC register islands, and the output fabric remaps and filters the traffic that leaves the SMC.

The input paths are shown in Figure 58, the local resource connections in Figure 59, and the outbound path in Figure 60. The three figures share the notation of the SEP fabric figures. Crossbars are rectangles, muxes and demuxes are trapezoids with the wide side towards the many ports, address remaps and rebases are hexagons and traffic filters are octagons. Green boxes are converters: address-width, data-width, ID-width and protocol conversion. Thick lines are

AXI4 and thin lines AXI-Lite; every AXI data path is 64 bits wide unless a converter narrows it. The labels give the address and ID widths where they change, and the port badges mark where a path crosses the module boundary.

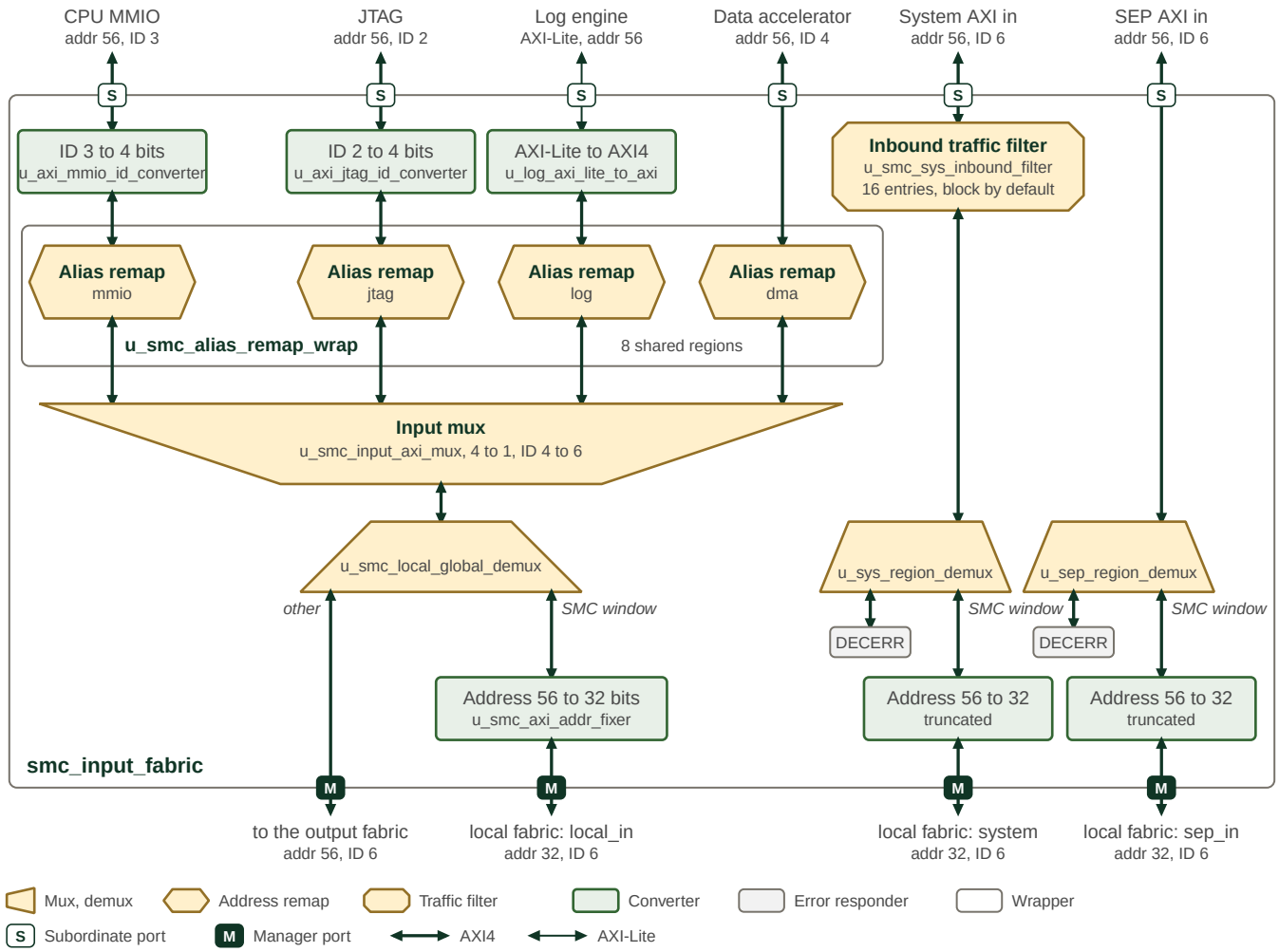


*Figure 58. SMC input fabric topology and routing*

Figure 58 has the six inputs of `smc_input_fabric` along the top and its four outputs along the bottom. The CPU MMIO and JTAG ports have their IDs widened to 4 bits and the log engine's AXI-Lite port is converted to AXI4. Each of the four SMC-internal initiators then passes its own alias remap; the four remaps share one table of eight regions with a 4 KiB granule. The input mux merges them, widens the ID to 6 bits and registers every channel. The demux behind it sends addresses inside the SMC window, at either the local or the global base and `REGION_SIZE` long, to the `local_in` initiator of the local fabric with the address truncated to 32 bits, and every other address to the output fabric. The system AXI input passes the inbound traffic filter, whose clock can be gated while the port is idle. The system and SEP inputs then each pass a region demux: the SMC window continues to the local fabric, truncated to 32 bits, and every other address receives DECERR. The SEP input is not filtered.

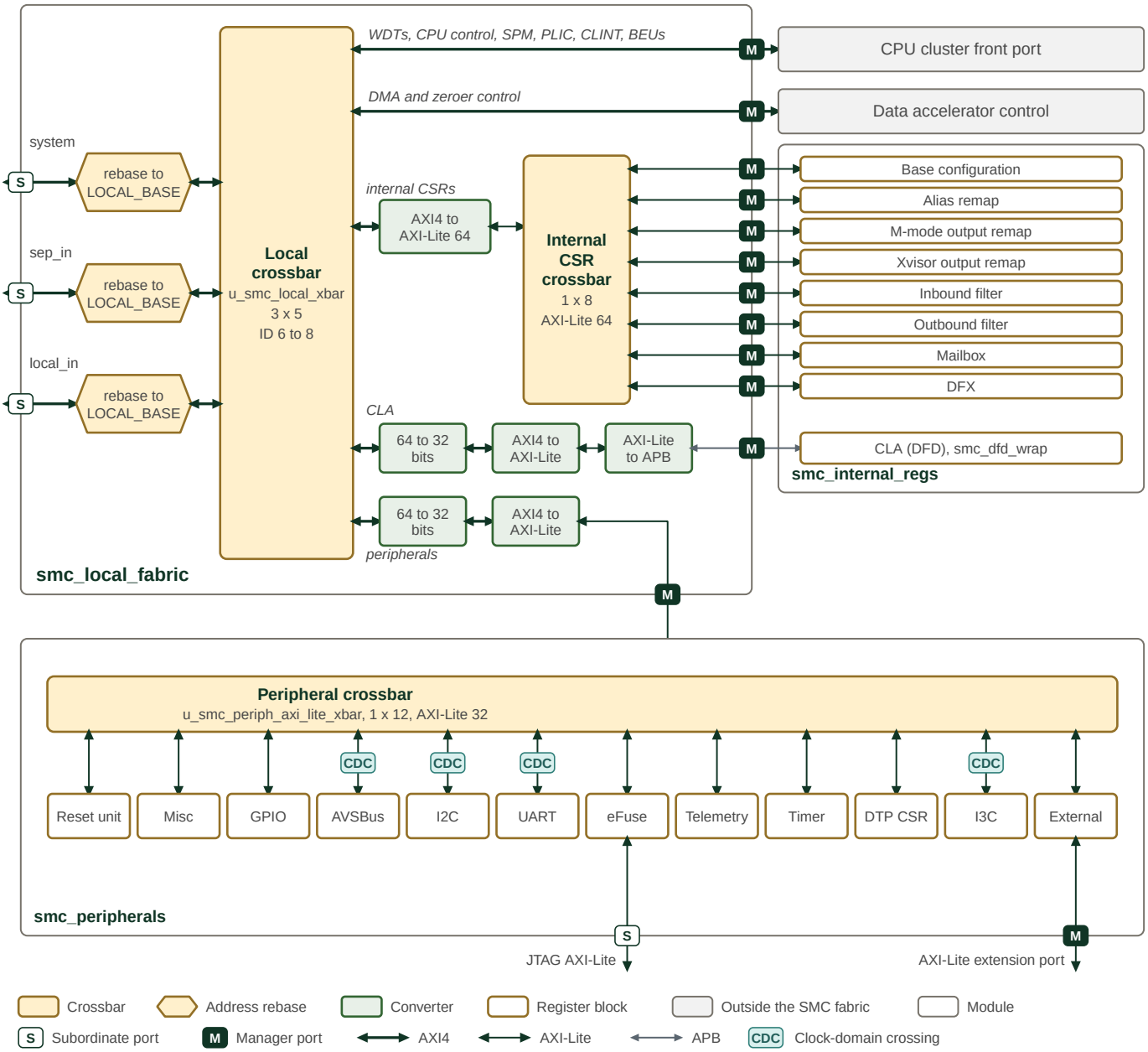


*Figure 59. SMC local fabric and peripheral islands*

Figure 59 shows `smc_local_fabric` in the upper left, the `smc_internal_regs` island on the right and the `smc_peripherals` island along the bottom. The three initiators arrive from the input fabric. Each is rebased onto the local base: the offset within `REGION_SIZE` is kept and the upper address bits are replaced by those of `LOCAL_BASE`, so a request for the global window reaches the same register as one for the local window. The local crossbar `u_smc_local_xbar` connects every initiator to every target, widens the ID to 8 bits, registers every port and answers unmapped addresses with DECERR, as do the internal CSR and peripheral crossbars. Its targets, in port order, are the CPU cluster front port, the data accelerator control port, the internal CSR crossbar, the peripheral crossbar and the CLA register window of the DFD block, which is reached over APB. The converters in front of the last three belong to `smc_local_xbar`. In the peripheral island, the AVSBus, I2C, UART and I3C register ports cross to the peripheral clock, the DTP CSR window is rebased to zero, and the eFuse registers have a second AXI-Lite port for JTAG access. The last peripheral crossbar target leaves the SMC as the AXI-Lite extension port.

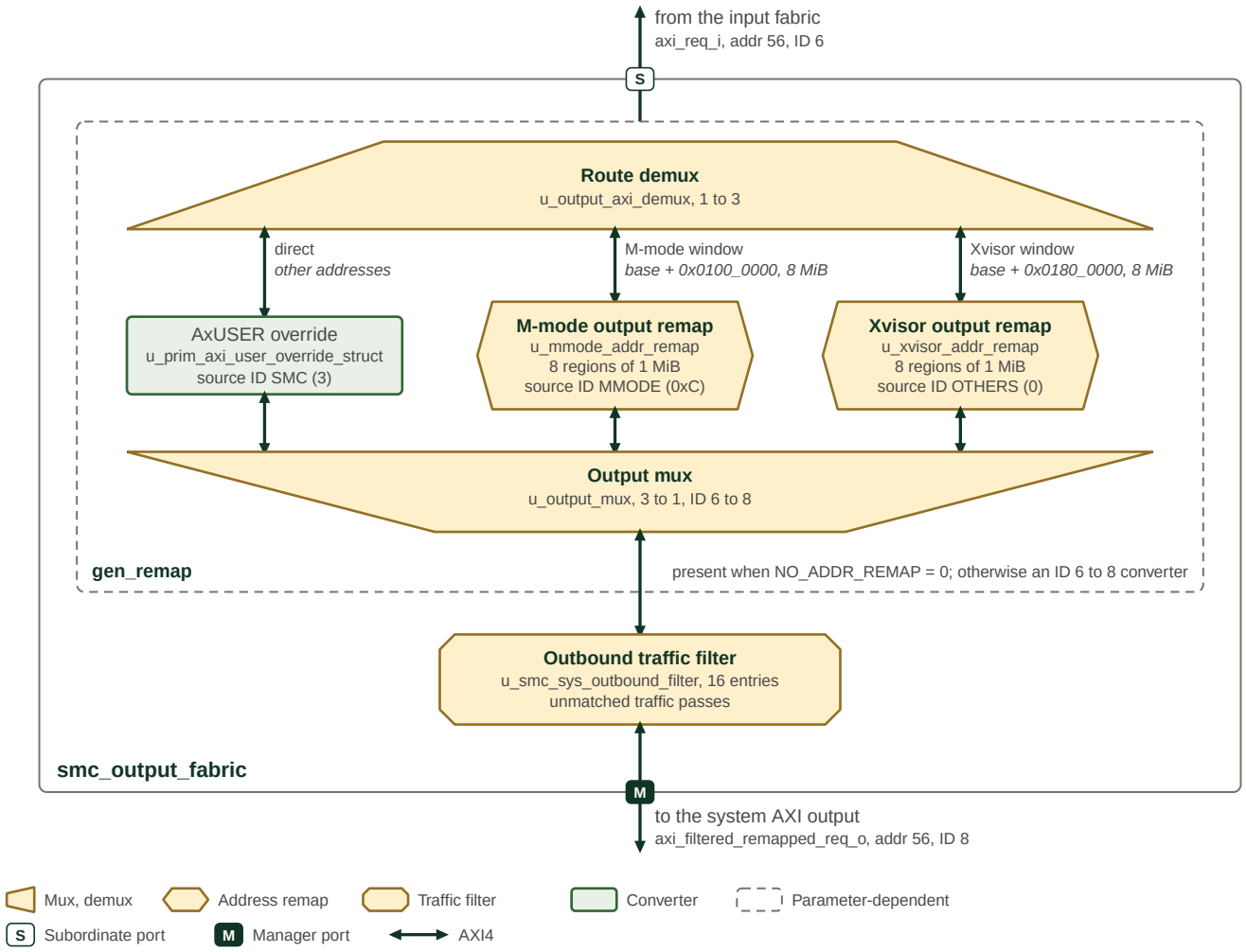


*Figure 60. SMC output fabric and remapping paths*

Figure 60 follows a request from the input fabric to the system AXI output. The route demux selects by address. The 8 MiB M-mode window at `0x0100_0000` and the 8 MiB Xvisor window at `0x0180_0000`, each above either the local or the global SMC base, go through their output remaps; every other address goes straight through. Each output remap divides its window into eight 1 MiB regions and replaces address bits [55:20] with the region's offset. Each path also sets the AxUSER source ID (Table 930). The output mux merges the three paths and widens the ID to 8 bits. The dashed outline marks the stage that the `NO_ADDR_REMAP` parameter removes: `smc_config_pkg` clears it, so the remap stage is present, and setting it leaves only an ID converter from 6 to 8 bits, without the source ID override. The outbound traffic filter passes traffic that matches no entry.

### Configuration Parameters

The fabric modules take parameters for the number of inbound and outbound filter entries, filter pipelining, the outstanding-transaction limit of the output fabric remap path, alias address remap enablement, and the M-mode and Xvisor remap region bases.

The parameters of `smc_fabric` are listed in the HTML edition of this manual, under RTL Modules Reference.

The parameters of `smc_output_fabric` are listed in the HTML edition of this manual, under RTL Modules Reference.

Further outstanding-transaction limits are fixed by package constants rather than module parameters (Table 923).

*Table 923. SMC Fabric Transaction Limits*

| Constant | Value | Description |
|---|---|---|
| **AXI4 Network** | | |
| `smc_pkg::FabricMaxTrans` | 32 | Maximum outstanding transactions per AXI ID bucket in the fabric demultiplexers and traffic filters |
| `smc_pkg::ErrSlvMaxTrans` | 32 | Maximum outstanding transactions for each error subordinate, across all IDs |
| `smc_local_xbar_pkg::XbarCfg.MaxMstTrans` | 8 | Maximum outstanding transactions each manager connected to the local AXI4 crossbar `u_smc_local_xbar` can have in flight |
| `smc_local_xbar_pkg::XbarCfg.MaxSlvTrans` | 8 | Maximum outstanding transactions each subordinate connected to the local AXI4 crossbar `u_smc_local_xbar` can have in flight |
| **AXI4-Lite Network** | | |
| `smc_internal_axi_lite_xbar_pkg::XbarCfg.MaxMstTrans` | 4 | Maximum outstanding transactions each manager connected to the local fabric AXI4-Lite crossbar can have in flight |
| `smc_internal_axi_lite_xbar_pkg::XbarCfg.MaxSlvTrans` | 4 | Maximum outstanding transactions each subordinate connected to the local fabric AXI4-Lite crossbar can have in flight |

These limits set the buffering depth of the fabric. The AXI4 limits bound the outstanding high-performance traffic, while the AXI4-Lite limits bound outstanding transactions to peripherals and configuration registers.

### Dual-Network Architecture

The SMC fabric implements two separate networks optimized for different traffic types (Table 924).

*Table 924. SMC Network Characteristics*

| Network | Connected Components | Protocol | Optimization Target |
|---|---|---|---|
| High-Performance | CPU cluster, SRAM, DMA, filtering, external AXI ports | AXI4 | Bandwidth, low latency |
| Low-Performance | Peripherals, sensors, configuration registers | AXI4-Lite | Reliability, deterministic access |

Both networks use 64-bit data bus width for consistent data handling. The CPU cluster connects to the fabric through AXI4 (high-bandwidth data path) and AXI4-Lite (MMIO) interfaces; internally, the cluster uses a TileLink bus that is protocol-bridged at the cluster boundary.

## Interfaces

### AXI Interface Widths

Table 925 lists the common AXI signal widths shared across all AXI4 fabric interfaces.

*Table 925. AXI Common Signal Widths*

| Signal | Width (bits) | Description |
|---|---|---|
| DATA_WIDTH | 64 | Data bus width for all AXI4 interfaces |
| STRB_WIDTH | 8 | Write strobe width (DATA_WIDTH / 8) |
| ADDR_WIDTH (system) | 56 | Address width on system-facing external ports |
| ADDR_WIDTH (local) | 32 | Address width on SMC-internal interfaces |
| USER_WIDTH | 12 | User sideband width for all AXI4 channels |

The AXI transaction ID width varies at each fabric stage because crossbars and multiplexers prepend additional bits to distinguish the originating port. Table 926 lists the ID width at each point in the fabric.

*Table 926. AXI ID Widths by Fabric Stage*

| Fabric Stage / Port | ID Width (bits) | Direction |
|---|---|---|
| System AXI input (sys_axi_in) | 6 | Inbound |
| JTAG AXI input (jtag_axi_in) | 2 | Inbound |
| SEP AXI input (sep_axi_in) | 6 | Inbound |
| Input fabric (slave side) | 4 | Internal |
| Local / Output fabric (slave side) | 6 | Internal |
| Local fabric crossbar (master side) | 8 | Internal |
| Output fabric (master side) | 8 | Outbound |
| System AXI output (output_axi) | 8 | Outbound |

Some internal paths also use a 32-bit data width for AXI4-Lite peripheral access, with corresponding STRB_WIDTH of 4 bits.

### Traffic Managers and Subordinates

The SMC fabric supports multiple traffic sources (managers) and destinations (subordinates), providing flexible routing with security enforcement.

### Traffic Managers

Traffic managers initiate transactions on the fabric, accessing both local and remote resources through the dual-network architecture.

Table 927 identifies the transaction initiators and their primary paths.

*Table 927. SMC Fabric Traffic Managers*

| Manager | Count | Primary Paths | Key Features |
|---|---|---|---|
| CPU Cluster | 1 | Local (SRAM, PLIC, CLINT, WDT), External (filtered/remapped) | Dual routing paths, privilege-based control, L2 coherent access |
| DMA Controller | 1 | Alias remap → filtering → destination | High-bandwidth data movement, security enforcement |
| JTAG2AXI Bridge | 1 | Alias remap → filtering → destination | Debug and test access, privilege control |
| Log Engine | 1 | Alias remap → filtering → destination | System logging, telemetry collection |

The CPU cluster provides multiple interface types: the local path optimized for minimal latency to local resources, the external path with filtering and remapping for system-wide access, the MMIO interface for register access with privilege-based control, and the L2 coherent path for cache-coherent high-bandwidth operations.

Traffic from DMA, Log Engine, JTAG2AXI, and CPU external path passes through alias remap logic before reaching their destination, enabling flexible address management between local and global address spaces.

### Traffic Subordinates

Traffic subordinates respond to transactions initiated by the managers, organized by network type and function.

Table 928 identifies the transaction targets and their network connections.

*Table 928. SMC Fabric Traffic Subordinates*

| Subordinate | Count | Network | Function |
|---|---|---|---|
| SRAM | 1 | AXI4 (High-Performance) | Local instruction and data memory |
| PLIC | 1 | AXI4 (High-Performance) | Platform-level interrupt controller |
| CLINT | 1 | AXI4 (High-Performance) | Core-local interrupt controller, timer |
| Watchdog Timer | 1 | AXI4 (High-Performance) | System integrity monitoring |
| Local Peripherals | Multiple | AXI4-Lite (Low-Performance) | SMC peripheral blocks (UART, I2C, GPIO, etc.) |
| Configuration Registers | Multiple | AXI4-Lite (Low-Performance) | Fabric control, filtering, remapping |
| External AXI Input Ports | 3 | AXI4 (High-Performance) | System NoC (sys_axi_in), JTAG debug (jtag_axi_in), SEP (sep_axi_in) |
| External AXI Output Port | 1 | AXI4 (High-Performance) | System NoC interface (output_axi) |
| Error Slave | 1 | Both networks | Invalid address response generation |

The subordinates are distributed across both networks based on their performance requirements and access patterns. High-performance subordinates (SRAM, PLIC, CLINT, WDT, external ports) connect to the AXI4 network for maximum bandwidth and minimal latency, while low-performance subordinates (peripherals, configuration registers) use the AXI4-Lite network for reliable, deterministic access.

### Address Remapping

The SMC fabric implements two complementary address remapping mechanisms: alias remapping in the input fabric and output remapping in the output fabric.

### Alias Remapping

Alias remapping translates local SMC addresses to global system addresses, decoupling software-visible addresses from physical assignments. This simplifies integration and software portability, letting local agents access resources consistently across different SoC mappings.

The alias remap logic organizes eight independently configurable regions, each specifying an input address range, a remap offset for translation, and an AxCACHE value that replaces the transaction's own on a hit. This enables:

- Translation between local alias addresses (0xC000_0000 at reset) and global addresses, ensuring compatibility with both internal and external software
- Support for multiple software images by allowing address regions to be reassigned for different execution instances
- Setting cacheable attributes on locally initiated transactions for optimal performance

At reset, remap logic defaults to transparent mapping (no remap), using the local alias base address (0xC000_0000) for local resources until programmed. The Programmer's Guide gives the aperture programming sequence.

### Output Remapping

Every outbound SMC request carries a source ID in AxUSER. The SMC outbound filter (Outbound Filtering) and the filters in front of other targets in the system match on it, together with address and NS, to decide what the request may reach.

The output remaps change that source ID. A request to a remap window is translated through the window's programmed regions and leaves with the window's source ID instead of the SMC's:

**M-mode remap**

`MMODE_ID`, the source ID of machine-mode firmware on another processor in the system.

**Xvisor remap**

`OTHER_ID`, the source ID of that processor's hypervisor and its guests.

**Direct path**

Any other address goes out untranslated, with `SMC_ID`.

The remaps are useful when the system integrates such a processor, for example an application processor (AP) running machine-mode firmware and the Xvisor hypervisor. SMC firmware can then reach memory the AP firmware owns through the M-mode window, or memory it shares with the hypervisor through the Xvisor window, and every filter treats the request as one from that software. A system without such a processor has no use for them.

M-mode and Xvisor name the other processor's software, not SMC privilege modes: SMC firmware runs in machine mode whichever window it uses. The address window alone selects the remap and the source ID; the privilege mode of the issuing SMC hart does not, and AxPROT, including NS, passes unchanged.

### Filtering and Protection

The filtering system (`filter_ctrl.rdl`) provides security control for both inbound and outbound traffic, examining transaction attributes to enforce access policies.

The SMC inbound and outbound filters are instances of the same `axi_filter_wrap` block used by SEP, and share its decode algorithm: entry priority, the match conditions, the address range granule and its widening direction, and the response returned to a blocked initiator. That algorithm is documented once with the IP.

See AXI Traffic Filter.

Both SMC filters have 16 entries, carry 56-bit addresses on a 64-bit data bus, and enable NS and source ID matching. They differ in how they treat traffic that matches no entry: the inbound filter blocks it, and the outbound filter permits it.

### Inbound Filtering

Figure 61 shows the decode that both filters implement. Their places in the fabric are in Figure 58 and Figure 60.

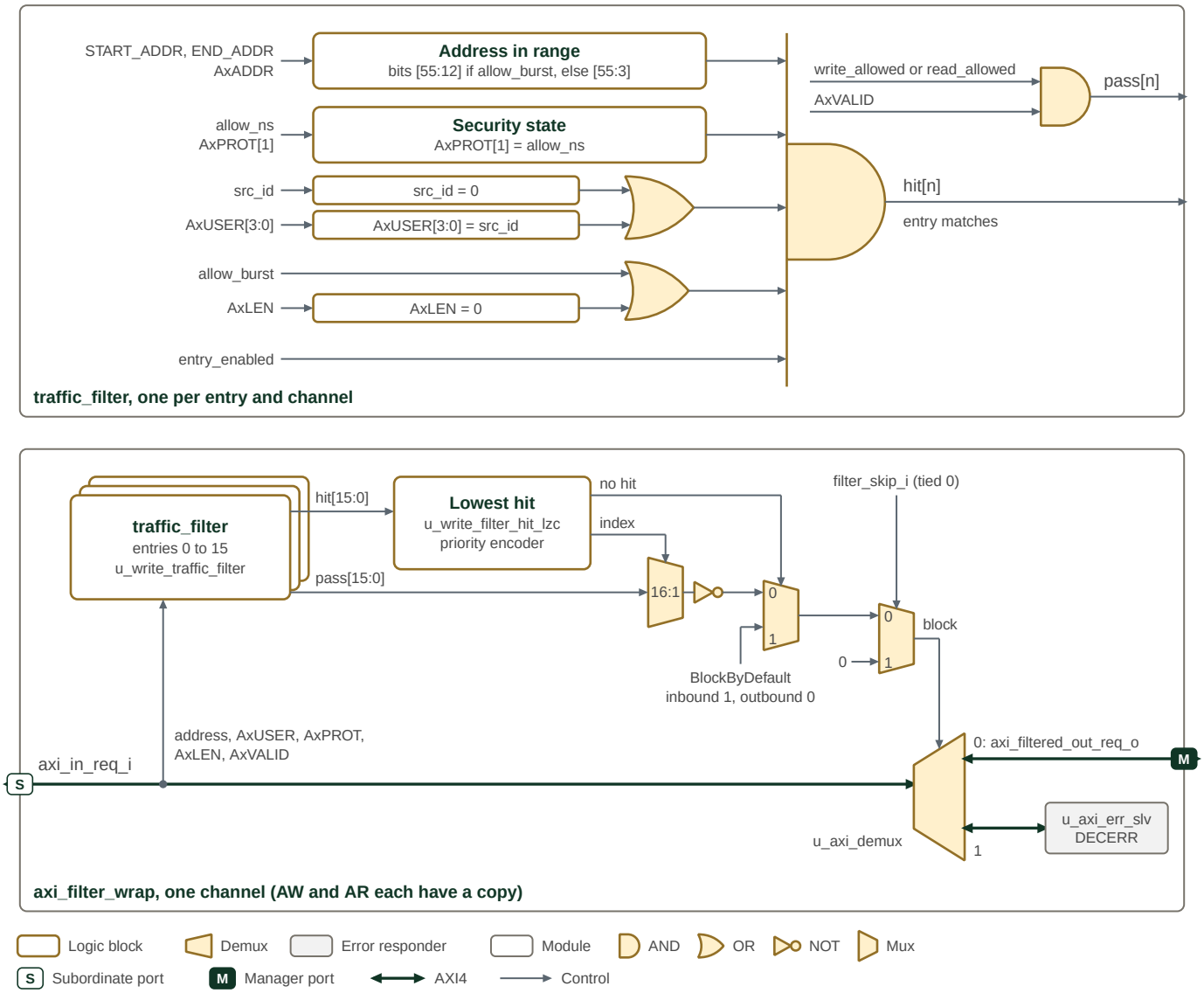


*Figure 61. SMC inbound and outbound traffic-filter mechanism*

The figure draws primitive logic with gate symbols and keeps labelled boxes for comparisons and larger blocks. The upper part of Figure 61 is one `traffic_filter`, instantiated for every entry on each of the AW and AR channels. The entry hits when all five conditions on the left hold, which the AND gate with the extended input bar combines; the source ID and burst conditions are each an OR of two terms. Its permission bit, ANDed with the channel's valid, is evaluated separately. The lower part is one channel of `axi_filter_wrap`. A priority encoder picks the lowest-numbered entry that hits, a 16:1 mux selects that entry's permission bit, and the inverted bit requests a block. When no entry hits, the next mux takes `BLOCK_BY_DEFAULT` instead. The last mux forces the block bit to 0 when `filter_skip_i` is set; `filter_skip_i` is tied low in both SMC instances. Group ID matching is not built into either instance. The block bit steers the transaction through a demux either to the filtered output or to an error responder that returns DECERR.

Inbound filtering protects SMC resources from unauthorized external access by evaluating three dimensions of each transaction:

- **Address-based control**: Defines precise access zones, protecting memory-mapped registers and internal memory from unauthorized external requests
- **Source ID authorization**: Permits or blocks accesses based on the originating system component's identity, ensuring only trusted sources access sensitive resources

- **Protection attribute filtering**: Examines AXI protection signals (security and privilege attributes) to differentiate between secure vs. non-secure or privileged vs. user transactions

The default filter configuration varies by chiplet type. Most SMC instances default to blocking all external transactions until access policies are configured. The Programmer's Guide gives the filter programming rules.

The inbound filter interface provides 16 independent filter instances, each configurable with its own address ranges, source ID matching, and protection attribute policies.

#### Outbound Filtering

The outbound filter uses explicit matching entries to allow or block SMC-initiated requests. Unmatched requests pass because this instance uses `BLOCK_BY_DEFAULT = 0`. An allow entry is therefore not an allowlist: denying traffic requires a matching entry whose read or write permission is clear.

Address, source ID, and NS matching select an entry. The filter does not infer software privilege or grant broader access to a higher privilege mode.

All outbound traffic, after passing through any output remap, is subject to outbound filtering enforcement based on address, security attributes (NS), and source ID. The outbound filters use the same register structure as inbound filters.

### Protection Requirements

The SMC fabric implements two types of protection-based filtering mechanisms to enforce security policies based on AXI protection attributes:

#### AXI Protection Bit Encoding

AXI protection bits (`aw.prot` and `ar.prot`) are 3-bit fields that encode transaction security and privilege attributes (Table 929).

*Table 929. SMC AXI Protection Bit Encoding*

| Bit [2] | Bit [1] | Bit [0] | Value | Description |
|---|---|---|---|---|
| 0 | 0 | 0 | 0x0 | Data, Secure, User |
| 0 | 0 | 1 | 0x1 | Data, Secure, Privileged |
| 0 | 1 | 0 | 0x2 | Data, Non-Secure, User |
| 0 | 1 | 1 | 0x3 | Data, Non-Secure, Privileged |
| 1 | 0 | 0 | 0x4 | Instruction, Secure, User |
| 1 | 0 | 1 | 0x5 | Instruction, Secure, Privileged |
| 1 | 1 | 0 | 0x6 | Instruction, Non-Secure, User |
| 1 | 1 | 1 | 0x7 | Instruction, Non-Secure, Privileged |

#### Non-Secure Bit Filtering (AXI4 Filters)

The primary protection filtering mechanism for AXI4 full interfaces examines the Non-Secure (NS) bit (`prot[1]`) to control secure vs. non-secure access. This filtering is implemented in both inbound and outbound traffic filters.

Each filter entry in the 16 inbound and 16 outbound filter instances includes an `allow_ns` configuration field. It is an equality test against the transaction's NS bit, not a permission grant, so an entry covers one security state only:

- When `allow_ns = 0`: the entry matches secure transactions (`prot[1] = 0`) only
- When `allow_ns = 1`: the entry matches non-secure transactions (`prot[1] = 1`) only

Covering both security states over one address range therefore takes two entries. An entry whose `allow_ns` does not match the transaction does not deny it; the transaction falls through to the remaining entries, as described in the AXI Traffic Filter decode section referenced above.

This filtering occurs in `axi_filter_wrap`, where each transaction's `aw.prot[1]` (write) or `ar.prot[1]` (read) is compared against the filter's `allow_ns` configuration. A matching deny entry routes the transaction to an error slave, which returns a decode error. With no matching entry, the inbound instance blocks the transaction and the outbound instance passes it.

#### Full 3-Bit Protection Matching (AXI-Lite)

The fabric filters examine only the NS bit. Exact matching on all three protection bits is performed by an AXI-Lite peripheral that provides its own `ACCESS_FILTER` register, after the fabric has routed the transaction to it. In the SMC, the GPIO interface bank performs this check; see GPIO. Adopter blocks in the external window that carry an `ACCESS_FILTER` register define the same fields.

The full protection matching requires an exact match between the transaction's protection bits and a configured requirement:

- `awprot_requirement[2:0]`: Required protection value for write transactions
- `arprot_requirement[2:0]`: Required protection value for read transactions
- `write_filter_enable`: Enable write protection filtering
- `read_filter_enable`: Enable read protection filtering

When filtering is enabled, only transactions with matching protection values are allowed through. All other transactions receive an error response. The protection bits are transaction attributes, not an initiator identity, so this check alone does not restrict a block to one initiator.

#### Protection Bit Pass-Through

All protection bits are passed through the fabric without modification. The filters only block or allow transactions based on their protection attributes; they do not alter the protection bits themselves. This ensures that downstream slaves receive the original protection attributes from the originating master.

#### Configuration Examples

The Programmer's Guide gives filter configuration examples, including secure-only and AXI-Lite PROT policies.

### Routing

#### Local and Remote Resource Access

The SMC fabric routing logic determines how transactions reach their intended destinations, supporting both local SMC resources and remote system components. Local resources can be addressed using either their global address or a local alias address.

The `GLOBAL_BASE`, `LOCAL_BASE`, and `REGION_SIZE` CSRs describe the SMC aperture. `REGION_SIZE` resets to `0x0100_0000` (16 MiB), and firmware may reprogram it during early boot to match the chiplet address assignment.

The SMC input fabric's local/global split is sized by the same `REGION_SIZE` CSR used by SMU-level routing and SEP-to-SMC decode, so the internal local-alias window and the chiplet-facing global aperture stay the same size by construction. A `REGION_SIZE` of zero collapses both windows and routes SMC CPU accesses to its own resources out through the output fabric.

After global-aperture reconfiguration, software may use global addresses for accesses that leave the local SMC view. Global address derivation requires the chiplet-global aperture base, which firmware reads from the `GLOBAL_BASE` CSR described above.

#### Outbound Traffic Flow

SMC outbound traffic serves dual purposes: accessing shared system services and extending the SMC's private domain across the system.

For resources that another processor's software owns or shares, such as shared memory, traffic passes through that software's output remap (Output Remapping). For private memory or memory-mapped I/O resources, traffic bypasses the remaps and goes directly to the system NoC, reducing latency.

Each outbound transaction carries a source ID naming the software it acts for, set by its traffic path (Table 930). A transaction that hits an alias remap region carries that region's AxCACHE value; any other keeps the AxCACHE its initiator issued.

*Table 930. SMC Source ID by Traffic Path*

| Traffic Path | Source ID | Acts For |
|---|---|---|
| Direct to NoC | SMC_ID | The SMC |
| Through Xvisor remap | OTHER_ID | Another processor's hypervisor and guests |
| Through M-mode remap | MMODE_ID | Another processor's machine-mode firmware |

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### Alias Remap Registers

The alias remap registers control local address aliasing and attribute mapping for SMC-initiated transactions. These registers configure how local resources are accessed and which address translation is applied.

#### AXI Alias Remap Address Map

TLB Registers for address remapping and attribute tagging

*Table 931. AXI Alias Remap Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | region_start | RW | - |
| 0x8 | region_end | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x10 | region_attrs | RW | - |

### region_start

*Table 932. alias_remap.region_start fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:12 | `start_addr` | RW | 0x0 | Start of remap region |
| 11:0 | `Reserved` | - | - | Reserved |

### region_end

*Table 933. alias_remap.region_end fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:12 | `end_addr` | RW | 0x0 | End of remap region (non-inclusive). |
| 11:0 | `Reserved` | - | - | Reserved |

### region_attrs

*Table 934. alias_remap.region_attrs fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `valid` | RW | 0x0 | If set, this remap will be active |
| 62:60 | `Reserved` | - | - | Reserved |
| 59:56 | `cacheable` | RW | 0x0 | Set to change the value of the axcache field of the transaction. This field will override all existing axcache bits. |
| 55:12 | `offset` | RW | 0x0 | The value of this field is added to bits [55:12] of the input address when it falls within the remap region. The lower 12 bits of the address are preserved unchanged. |
| 11:0 | `Reserved` | - | - | Reserved |

### Output Remap Registers

The output remap registers program the M-mode and Xvisor output remap regions (Output Remapping). Each region carries a `REGION_ATTRS.valid` bit (reset 0): when set, accesses selecting the region are remapped through its offset; when clear, they pass through with their address unchanged, so an unprogrammed entry acts as an identity mapping rather than forcing every access to offset 0.

### Output Remap Address Map

Address-translation registers that remap outbound transactions and tag their attributes.

*Table 935. Output Remap Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | region_attrs | RW | - |

### region_attrs

*Table 936. output_remap.region_attrs fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `valid` | RW | 0x0 | If set, accesses to this region are remapped by offset; if clear, they pass through with their address unchanged |
| 62:56 | `Reserved` | - | - | Reserved |
| 55:0 | `offset` | RW | 0x0 | Address translation for remap region. RTL will use the number of bits appropriate for region granulaity. e.g. SMC: 8 * 1MB regions -→ 20 bits of address space per region, 36 MSBs from this register used SEP: 16 * 512KB regions -→ 19 bits of address space per region, 37 MSBs from this register used |

### Inbound Traffic Filter Control Registers

The inbound filter control registers protect SMC resources from unauthorized external access. Each of the 16 independent filter instances can be configured with address ranges, source ID matching, and protection attribute policies to control inbound traffic.

### AXI Filter Control Address Map

Per-filter access-control registers defining permitted address ranges and transaction attributes.

*Table 937. AXI Filter Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | FILTER_CONFIG | RW | - |
| 0x8 | START_ADDR | RW | - |
| 0x10 | END_ADDR | RW | - |

### FILTER_CONFIG

*Table 938. filter_ctrl.FILTER_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `locked` | RW1S | 0x0 | Lock filter configurations. Write once register. Once set, this bit cannot be cleared, and every subsequent write to this filter entry's FILTER_CONFIG, START_ADDR and END_ADDR registers is steered to the AXI error subordinate and terminates with a DECERR response; reads still return the locked configuration. |
| 62:25 | `Reserved` | - | - | Reserved |
| 24 | `allow_burst` | RW | 0x0 | Filter entry will match if this field is 1 or if this field is 0 and the burst len AxLEN is 0. This field also selects the granule of the address range: 4KB when it is 1, 2^data_bus_width bytes when it is 0. See start_addr and end_addr for how the range is widened to that granule. |
| 23:20 | `group_id` | RW | 0x0 | Group ID field to match. Filter entry will match if the transactions group_id matches the value in this field. Note the corner case is the opposite of src_id: the bypass keys off the transaction, not this field. A transaction whose group_id is 0 matches regardless of this field, and a value of 0 here is not a wildcard. Unused for SEP/SMC, where group_id filtering is disabled at build time and this field has no effect |
| 19:16 | `src_id` | RW | 0x0 | Source ID field to match. Filter entry will match if the transactions src_id matches the value in this field. A value of 0 in this field is a wildcard: src_id matching is bypassed and the entry matches any src_id. Consequently there is no way to match only transactions whose src_id is 0. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15 | `Reserved` | - | - | Reserved |
| 14:12 | `data_bus_width` | R | 0x3 | Data bus width in bytes, encoded as 2^value |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `allow_ns` | RW | 0x0 | Secure and Non-secure is categorized as AxPROT[1] in the AXI protocol, where AxPROT[1] == 0 means secure and AxPROT[1] == 1 means Non-secure. Filter entry will match if AxPROT[1] matches the value in this field |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `entry_enabled` | RW | 0x0 | Filter entry enabled. A = 0 means this entry is disabled, A = 1 means this entry is enabled and will be used to filter |
| 3:2 | `Reserved` | - | - | Reserved |
| 1 | `write_allowed` | RW | 0x0 | Upon match of transaction to filter fields, whether the transaction has write permissions |
| 0 | `read_allowed` | RW | 0x0 | Upon match of transaction to filter fields, whether the transaction has read permissions |

#### START_ADDR

*Table 939. filter_ctrl.START_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `start_addr` | RW | 0x0 | Start of the allowable address range, rounded down to the base of its granule. The granule is 4KB when allow_burst is 1, and 2^data_bus_width bytes when allow_burst is 0. Address matching ignores the address bits below the granule, so the accepted range always spans whole granules and is never narrower than one: a range shorter than the granule, start_addr == end_addr included, still admits the whole granule, and two granules if it straddles a granule boundary. Hardware writes the rounded start_addr and end_addr back into these fields only when both land in the same granule; otherwise readback returns the programmed values even though matching is still granule-aligned. |

#### END_ADDR

*Table 940. filter_ctrl.END_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `end_addr` | RW | 0x7 | End of the allowable address range, rounded up to the top of its granule. The granule and the write-back behaviour are described under start_addr. Defaults to 7 on reset to match the 8B granule that applies while bursts are disabled. |

### Outbound Traffic Filter Control Registers

The outbound filter control registers determine which external resources the SMC can access, restricting SMC-initiated requests to defined system peripherals or memory regions. These registers are implemented as 16 independent outbound filter instances, each structured as a 32-byte control block and accessible via an AXI4-Lite interface. The outbound filters use the same register structure as the inbound filters (documented above).

### 2.1.6.6. Interrupts

SMC external interrupts pass through the Platform-Level Interrupt Controller (PLIC). CLINT supplies per-core timer and software interrupts. Each bus error unit (BEU) has both a direct core output and a separately selected PLIC output. This chapter distinguishes raw SMC vector bits from software-visible PLIC IDs.

#### Controllers

#### Controller Addresses

These addresses use the SMC-local view. N ranges from 0 through 3 for the per-core watchdog and BEU apertures.

*Table 941. SMC interrupt-controller addresses*

| Block | Base Addr | Size | Interface |
|---|---|---|---|
| CPU Control Regs | 0xC003_9000 | 4 KB | AXI4-Lite |
| PLIC | 0xC400_0000 | 64 MB | TileLink (cluster control bus) |
| CLINT | 0xC800_0000 | 64 KB | TileLink (cluster control bus) |
| Watchdog Timers (N) | 0xC000_0000 + (N * 0x400) | 1 KB x4 | TileLink (cluster peripheral bus) |
| Bus Error Units (N) | 0xC801_0000 + (N * 0x1000) | 4 KB x4 | TileLink (cluster control bus) |

#### PLIC Sources and Contexts

The four-core implementation has 336 PLIC inputs with source IDs 1–336. ID 0 means no interrupt. The inputs comprise:

*Table 942. PLIC input groups*

| Input group | PLIC IDs | Origin |
|---|---|---|
| Raw SMC vector bits 0–327 | 1–328 | `cpu_interrupts_o`; detailed below. |
| Cluster watchdogs 0–3 | 329–332 | Added by the cluster interrupt crossbar. |
| Core BEU PLIC outputs 0–3 | 333–336 | Added within the PLIC clock domain. |

Reserved raw-vector positions remain reserved PLIC IDs. The 256 external input meanings come from the enclosing SoC integration; an input count is not a promise that every position is an active source.

There are eight delivery contexts: machine contexts for cores 0–3, then supervisor contexts for cores 0–3. Priorities are shared per source; enable masks, thresholds and claim/complete registers are per context. A priority must exceed the context threshold to raise its external-interrupt output. Priority zero suppresses a source. See the generated map for context offsets and priority range. These settings route interrupts; they do not themselves enforce MMIO access policy.

The gateways are active-high and level-sensitive, with no selectable edge mode. External SMC pins pass through `prim_sync3` before the raw vector. A short pulse can be missed if it is not sampled; stretch it or hold the request until serviced. A gateway admits one request and waits for completion before accepting another. It does not count repeated pulses while one request is in flight.

#### CLINT

CLINT provides a shared 64-bit timer, per-core compare registers and per-core software-interrupt state. Timer and software interrupts go directly to the cores rather than through PLIC claim/complete. The timer advances on the

cluster's periodic tick; see the Clocks and Resets chapter for its clock relationship.

#### Bus Error Units

A Bus Error Unit (BEU) accumulates error events for one core. The register named `ACCRUED_ENABLE` is sticky status, not an enable mask: events set its bits independently of `ENABLE`. Ordinary writes replace those bits; write zero to clear them. `ENABLE` instead gates capture of the first error's `CAUSE` and `PHYS_ADDR` while `CAUSE` is zero. Software can also write `PHYS_ADDR`.

`LOCAL_ENABLE` selects accrued errors for direct delivery to the core, bypassing the PLIC and the ordinary per-source `mie` bits. This is not an unconditionally non-maskable interrupt: in machine mode, `mstatus.MIE = 0` suppresses delivery. The machine interrupt is enabled while executing in lower privilege modes. The BEU also has a separate output selected by `PLIC_ENABLE`. The Bus Error Unit service sequence is in the Programmer's Guide.

### Sources and Routing

#### Raw Vector and PLIC Numbering

Raw bit N maps to PLIC source ID N+1. For example, DMA completion is raw bit 322 and PLIC ID 323. Watchdog IDs 329–332 and BEU IDs 333–336 are not additional bits on the 328-bit raw SMC vector.

#### SMC CPU Interrupt Vector Map

This section documents the raw `cpu_interrupts_o` vector assembled in `hw/sys/smc/rtl/smc_base.sv`. The vector is built from four sources:

- `smc_ext_interrupts_i`
- `peripheral_interrupts_i[31:0]`
- `mailbox_interrupts[31:0]`
- four internal interrupt signals

Unless otherwise noted, the bit numbers below are raw `cpu_interrupts_o` bit indices, not PLIC source IDs. When these signals feed the PLIC, the PLIC source ID is `cpu_interrupts_o` bit index + 1.

The vector layout is:

```
cpu_interrupts_o[NumExtInterrupts-1:0]        = ext_interrupts_smc_clk; //
smc_ext_interrupts_i after 3-stage sync to clk_smc
cpu_interrupts_o[NumExtInterrupts+:32]        = peripheral_interrupts_i;
cpu_interrupts_o[(NumExtInterrupts+32)+:32]   = mailbox_interrupts;
cpu_interrupts_o[(NumExtInterrupts+64)]       = tdr_dbg_ctrl_clocks_stopped_by_cla_o;
cpu_interrupts_o[(NumExtInterrupts+64)+1]     = cla_interrupt;
cpu_interrupts_o[(NumExtInterrupts+64)+2]     = dma_intp;
cpu_interrupts_o[(NumExtInterrupts+64)+3]     = zeroer_intp;
```

`NumExtInterrupts = 256` and `NumCpuInterrupts = 328`.

### Exact Indexed Map

Table 943 shows the exact raw `cpu_interrupts_o` indexing. The `Source` column is intentionally human-readable; implementation signal names are called out in the `RTL trace` column instead.

*Table 943. SMC indexed interrupt map*

| Bit(s) | Source | Description | RTL trace |
|---|---|---|---|
| `[255:0]` | External interrupts | Integration-defined external interrupt inputs | `hw/sys/smc/rtl/smc_base.sv` synchronizes `ext_interrupts_i` to `clk_smc` (`prim_sync3 u_ext_interrupts_sync3`; wired from `smc.sv smc_ext_interrupts_i`) and drives `cpu_interrupts_o[NumExtInterrupts-1:0] = ext_interrupts_smc_clk`. |
| `256` | SEP mailbox interrupt 0 | SEP-to-SMC mailbox channel 0 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[0] = sep_mailbox_interrupts_i[0]`. |
| `257` | SEP mailbox interrupt 1 | SEP-to-SMC mailbox channel 1 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[1] = sep_mailbox_interrupts_i[1]`. |
| `258` | SEP mailbox interrupt 2 | SEP-to-SMC mailbox channel 2 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[2] = sep_mailbox_interrupts_i[2]`. |
| `259` | SEP mailbox interrupt 3 | SEP-to-SMC mailbox channel 3 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[3] = sep_mailbox_interrupts_i[3]`. |
| `260` | SEP mailbox interrupt 4 | SEP-to-SMC mailbox channel 4 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[4] = sep_mailbox_interrupts_i[4]`. |
| `261` | SEP mailbox interrupt 5 | SEP-to-SMC mailbox channel 5 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[5] = sep_mailbox_interrupts_i[5]`. |
| `262` | SEP mailbox interrupt 6 | SEP-to-SMC mailbox channel 6 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[6] = sep_mailbox_interrupts_i[6]`. |
| `263` | SEP mailbox interrupt 7 | SEP-to-SMC mailbox channel 7 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[7] = sep_mailbox_interrupts_i[7]`. |
| `264` | Telemetry interrupt 0 | Telemetry receiver instance 0 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[8] = telemetry_irq[0]`. |
| `265` | Telemetry interrupt 1 | Telemetry receiver instance 1 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[9] = telemetry_irq[1]`. |
| `266` | Telemetry interrupt 2 | Telemetry receiver instance 2 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[10] = telemetry_irq[2]`. |
| `267` | NDM Reset Request | OR-reduced synchronized NDM reset request interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[11] = \|ndmreset_request_smc_clk`. |
| `268` | I3C interrupt 0 | I3C controller instance 0 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[12] = i3c_irqs_smc_clk[0]`; sourced from the I3C wrapper through the peripheral CDC block. |
| `269` | I3C interrupt 1 | I3C controller instance 1 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[13] = i3c_irqs_smc_clk[1]`; sourced from the I3C wrapper through the peripheral CDC block. |
| `270` | I3C interrupt 2 | I3C controller instance 2 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[14] = i3c_irqs_smc_clk[2]`; sourced from the I3C wrapper through the peripheral CDC block. |
| `271` | I3C interrupt 3 | I3C controller instance 3 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[15] = i3c_irqs_smc_clk[3]`; sourced from the I3C wrapper through the peripheral CDC block. |
| `272` | I3C interrupt 4 | I3C controller instance 4 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[16] = i3c_irqs_smc_clk[4]`; sourced from the I3C wrapper through the peripheral CDC block. |
| `273` | I3C interrupt 5 | I3C controller instance 5 interrupt | `smc_peripherals.sv` drives `peripheral_interrupts_o[17] = i3c_irqs_smc_clk[5]`; sourced from the I3C wrapper through the peripheral CDC block. |

| Bit(s) | Source | Description | RTL trace |
|---|---|---|---|
| 274 | UART/Log Engine interrupt 0 | Combined UART IRQ, UART error, and log-engine interrupt for instance 0 | smc_peripherals.sv drives peripheral_interrupts_o[18] = uart_irq_combined_smc_clk[0]; the combined signal is formed in smc_peripherals_cdc.sv. |
| 275 | UART/Log Engine interrupt 1 | Combined UART IRQ, UART error, and log-engine interrupt for instance 1 | smc_peripherals.sv drives peripheral_interrupts_o[19] = uart_irq_combined_smc_clk[1]; the combined signal is formed in smc_peripherals_cdc.sv. |
| 276 | UART/Log Engine interrupt 2 | Combined UART IRQ, UART error, and log-engine interrupt for instance 2 | smc_peripherals.sv drives peripheral_interrupts_o[20] = uart_irq_combined_smc_clk[2]; the combined signal is formed in smc_peripherals_cdc.sv. |
| 277 | UART/Log Engine interrupt 3 | Combined UART IRQ, UART error, and log-engine interrupt for instance 3 | smc_peripherals.sv drives peripheral_interrupts_o[21] = uart_irq_combined_smc_clk[3]; the combined signal is formed in smc_peripherals_cdc.sv. |
| 278 | AVSBus interrupt | Adaptive voltage scaling interrupt | smc_peripherals.sv drives peripheral_interrupts_o[22] = avsbus_irq_smc_clk. |
| 279 | I2C interrupt 0 | I2C controller instance 0 interrupt | smc_peripherals.sv drives peripheral_interrupts_o[23] = i2c_irqs_smc_clk[0]. |
| 280 | I2C interrupt 1 | I2C controller instance 1 interrupt | smc_peripherals.sv drives peripheral_interrupts_o[24] = i2c_irqs_smc_clk[1]. |
| 281 | I2C interrupt 2 | I2C controller instance 2 interrupt | smc_peripherals.sv drives peripheral_interrupts_o[25] = i2c_irqs_smc_clk[2]. |
| 282 | SEP watchdog reset | External watchdog reset indication from SEP path | smc_peripherals.sv synchronizes ~rst_ext_wdt_ni through u_rst_ext_wdt_irq_sync and drives peripheral_interrupts_o[26] = rst_ext_wdt_smc_clk. |
| 283 | Locked field access | eFuse locked-field access violation interrupt | smc_peripherals.sv drives peripheral_interrupts_o[27] = locked_field_access_interrupt, sourced from the eFuse controller path. |
| 284 | GPIO interrupt OR, lower half | OR-reduction across GPIO interrupt sources 0-31 | smc_peripherals.sv drives peripheral_interrupts_o[28] = \|gpio_interrupt[smc_pkg::NumGpioWraps/2-1:0]. |
| 285 | GPIO interrupt OR, upper half | OR-reduction across GPIO interrupt sources 32-64 | smc_peripherals.sv drives peripheral_interrupts_o[29] = \|gpio_interrupt[smc_pkg::NumGpioWraps-1:smc_pkg::NumGpioWraps/2]. |
| 286 | AXI hang detector OR | OR of the three AXI hang detectors (sys_axi, sep_axi, data_accel). Asserts while any detector's irq_o is high, either from a timeout or from its irq_test bit, so software must read SMC_BASE_CONFIG.HANG_DET_{SYS_AXI,SEP_AXI,DATA_ACCEL}_CTRL.irq to identify the stalled master. There is no status latch to clear: a timeout drops when that master's bus makes progress, a test assertion drops when irq_test is cleared, and either drops when the detector's enable or irq_en is cleared. The CPU AXI master is not covered here; it is covered by the watchdog. | smc_base.sv OR-reduces hang_irq_sys_axi \| hang_irq_sep_axi \| hang_irq_data_accel onto axi_hang_irq_o; smc.sv routes that back into u_smc_peripherals, and smc_peripherals.sv drives peripheral_interrupts_o[30] = axi_hang_irq_i. |
| 287 | Reserved | Unused peripheral interrupt slots; tied to zero | peripheral_interrupts_o is initialized to '0, and bit 31 is never overwritten in smc_peripherals.sv. |
| [319:288] | SMC mailbox interrupts [31:0] | Inbound SMC mailbox interrupt channels 31:0 | smc_base.sv drives cpu_interrupts_o[(NumExtInterrupts+32)+:32] = mailbox_interrupts; mailbox.sv drives inbound_interrupt_o[m] per mailbox channel. |
| 320 | CLA Clock Stop | DFD CLA clock-stop status interrupt | smc_base.sv drives this bit from tdr_dbg_ctrl_clocks_stopped_by_cla_o, sourced through u_smc_dfd_wrap in smc_internal_regs.sv. |

| Bit(s) | Source | Description | RTL trace |
|---|---|---|---|
| 321 | CLA Interrupt | DFD CLA interrupt | smc_base.sv drives this bit from cla_interrupt, sourced from external_action_debug_interrupt_o on u_smc_dfd_wrap in smc_internal_regs.sv. |
| 322 | DMA transfer complete | Falling-edge pulse when DMA busy deasserts | smc_base.sv drives this bit from dma_intp; idma_wrapper.sv generates dma_intp_o as a falling-edge pulse of dma_busy_o. |
| 323 | Zeroer transfer complete | Falling-edge pulse when Zeroer busy deasserts | smc_base.sv drives this bit from zeroer_intp; zeroer.sv generates zeroer_intp_o on the falling edge of zeroer_busy_o. |
| [327:324] | Reserved | Unused tail bits; tied to zero | cpu_interrupts_o is initialized to '0, and no RTL overwrites these tail bits in smc_base.sv. |

#### Routing Notes

smc_base.sv assembles the raw vector; smc_peripherals.sv assembles its 32-bit peripheral slice. The AXI hang-detector OR enters peripheral bit 30 (raw bit 286), while bit 31 is reserved. The table preserves the individual sources behind combined GPIO, UART/log-engine and hang-detector indications; software must inspect their local status to identify and service the cause.

The six I3C entries are connected through the peripheral CDC block. The AVSBus and external-watchdog indications are also synchronized into the SMC domain. Do not infer pulse-capture guarantees merely from the presence of a synchronizer. DMA and zeroer completion indications are busy-falling pulses, not counted queues.

#### Software Handling

The PLIC priority, enable and threshold registers are not reset by hardware. A source's priority is shared by every context that enables it. Completion releases the source's gateway, so a still-asserted source pends again; claim and complete do not clear a peripheral's own sticky status. CLINT timer and software interrupts do not use PLIC completion.

The Interrupt Control section of the Programmer's Guide gives the PLIC initialization and claim/complete sequence, CLINT timer and software interrupt service, and Bus Error Unit service. Watchdog recovery and CPU reset/isolation ordering are described in the Watchdog Timers and CPU chapters. For CPU cluster context, see [cpu-memory-system].

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### PLIC Address Map

Platform-Level Interrupt Controller register interface.

**NOTE**

**Register Arrays:**

- **PRIORITY[337]**: 337 registers, base 0x0, stride 0x4
- **PENDING[11]**: 11 registers, base 0x1000, stride 0x4
- **CORE0_MEIP_ENABLE[11]**: 11 registers, base 0x2000, stride 0x4
- **CORE1_MEIP_ENABLE[11]**: 11 registers, base 0x2080, stride 0x4

- **CORE2_MEIP_ENABLE[11]**: 11 registers, base 0x2100, stride 0x4
- **CORE3_MEIP_ENABLE[11]**: 11 registers, base 0x2180, stride 0x4
- **CORE0_SEIP_ENABLE[11]**: 11 registers, base 0x2200, stride 0x4
- **CORE1_SEIP_ENABLE[11]**: 11 registers, base 0x2280, stride 0x4
- **CORE2_SEIP_ENABLE[11]**: 11 registers, base 0x2300, stride 0x4
- **CORE3_SEIP_ENABLE[11]**: 11 registers, base 0x2380, stride 0x4

*Table 944. PLIC Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x540 | PRIORITY[337] | RW | - |
| 0x1000 - 0x1028 | PENDING[11] | R | - |
| 0x2000 - 0x2028 | CORE0_MEIP_ENABLE[11] | RW | - |
| 0x2080 - 0x20A8 | CORE1_MEIP_ENABLE[11] | RW | - |
| 0x2100 - 0x2128 | CORE2_MEIP_ENABLE[11] | RW | - |
| 0x2180 - 0x21A8 | CORE3_MEIP_ENABLE[11] | RW | - |
| 0x2200 - 0x2228 | CORE0_SEIP_ENABLE[11] | RW | - |
| 0x2280 - 0x22A8 | CORE1_SEIP_ENABLE[11] | RW | - |
| 0x2300 - 0x2328 | CORE2_SEIP_ENABLE[11] | RW | - |
| 0x2380 - 0x23A8 | CORE3_SEIP_ENABLE[11] | RW | - |
| 0x200000 | CORE0_MEIP_THRESHOLD | RW | - |
| 0x200004 | CORE0_MEIP_CLAIM_COMPLETE | RW | - |
| 0x201000 | CORE1_MEIP_THRESHOLD | RW | - |
| 0x201004 | CORE1_MEIP_CLAIM_COMPLETE | RW | - |
| 0x202000 | CORE2_MEIP_THRESHOLD | RW | - |
| 0x202004 | CORE2_MEIP_CLAIM_COMPLETE | RW | - |
| 0x203000 | CORE3_MEIP_THRESHOLD | RW | - |
| 0x203004 | CORE3_MEIP_CLAIM_COMPLETE | RW | - |
| 0x204000 | CORE0_SEIP_THRESHOLD | RW | - |
| 0x204004 | CORE0_SEIP_CLAIM_COMPLETE | RW | - |
| 0x205000 | CORE1_SEIP_THRESHOLD | RW | - |
| 0x205004 | CORE1_SEIP_CLAIM_COMPLETE | RW | - |
| 0x206000 | CORE2_SEIP_THRESHOLD | RW | - |
| 0x206004 | CORE2_SEIP_CLAIM_COMPLETE | RW | - |
| 0x207000 | CORE3_SEIP_THRESHOLD | RW | - |
| 0x207004 | CORE3_SEIP_CLAIM_COMPLETE | RW | - |

#### PRIORITY[337]

*Table 945. plic.PRIORITY[337] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Priority of interrupt. The higher the set value, the higher the priority. For interrupts with the same priority, the tie is broken by interrupt ID; interrupts with the lowest ID has highest effective priority. Interrupts start counting at 1. |

#### PENDING[11]

*Table 946. plic.PENDING[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | R | 0x0 | Set to 1 if interrupt source 1 is pending, regardless of its enable or priority setting. Interrupts start counting at 1. |

#### CORE0_MEIP_ENABLE[11]

*Table 947. plic.CORE0_MEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

#### CORE1_MEIP_ENABLE[11]

*Table 948. plic.CORE1_MEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

#### CORE2_MEIP_ENABLE[11]

*Table 949. plic.CORE2_MEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

#### CORE3_MEIP_ENABLE[11]

*Table 950. plic.CORE3_MEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

#### CORE0_SEIP_ENABLE[11]

*Table 951. plic.CORE0_SEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

### CORE1_SEIP_ENABLE[11]

*Table 952. plic.CORE1_SEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

### CORE2_SEIP_ENABLE[11]

*Table 953. plic.CORE2_SEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

### CORE3_SEIP_ENABLE[11]

*Table 954. plic.CORE3_SEIP_ENABLE[11] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Set bits to 1 if interrupt should be enabled. Interrupts start counting at 1. |

### CORE0_MEIP_THRESHOLD

*Table 955. plic.CORE0_MEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE0_MEIP_CLAIM_COMPLETE

*Table 956. plic.CORE0_MEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE1_MEIP_THRESHOLD

*Table 957. plic.CORE1_MEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE1_MEIP_CLAIM_COMPLETE

*Table 958. plic.CORE1_MEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE2_MEIP_THRESHOLD

*Table 959. plic.CORE2_MEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE2_MEIP_CLAIM_COMPLETE

*Table 960. plic.CORE2_MEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE3_MEIP_THRESHOLD

*Table 961. plic.CORE3_MEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE3_MEIP_CLAIM_COMPLETE

*Table 962. plic.CORE3_MEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE0_SEIP_THRESHOLD

*Table 963. plic.CORE0_SEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE0_SEIP_CLAIM_COMPLETE

*Table 964. plic.CORE0_SEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE1_SEIP_THRESHOLD

*Table 965. plic.CORE1_SEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE1_SEIP_CLAIM_COMPLETE

*Table 966. plic.CORE1_SEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### CORE2_SEIP_THRESHOLD

*Table 967. plic.CORE2_SEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

### CORE2_SEIP_CLAIM_COMPLETE

*Table 968. plic.CORE2_SEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

#### CORE3_SEIP_THRESHOLD

*Table 969. plic.CORE3_SEIP_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2:0 | `value` | RW | - | Interrupt notification threshold. Maximum value is 7. The context's external interrupt is raised only while the highest-priority pending, enabled source has a priority strictly greater than the threshold. The threshold does not gate claims: a claim read returns the highest-priority pending, enabled source regardless of the threshold. |

#### CORE3_SEIP_CLAIM_COMPLETE

*Table 970. plic.CORE3_SEIP_CLAIM_COMPLETE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `value` | RW | - | Claim/Complete register. Reading this register returns the claimed interrupt number and makes it no longer pending. Writing the interrupt number back completes the interrupt. |

### 2.1.6.7. Peripherals

This section documents the main peripherals integrated within the System Management Controller (SMC). These peripherals provide essential I/O and communication capabilities for the OCAH system, supporting various protocols and interfaces required for system operation and external connectivity.

Table 971 summarizes the key peripherals, their communication protocols, and default instance counts:

*Table 971. SMC Peripheral Summary*

| Peripheral | Description | Protocol | Instances |
|---|---|---|---|
| Mailbox | FIFO-based bidirectional communication channels for inter-processor or inter-chiplet communication. | AXI4-Lite | 32 |
| System Timer OCTS | 64-bit timer for multi-chiplet time synchronization with PRIMARY/SECONDARY modes. | APB4/AXI4-Lite | 1 |
| eFuse | Secure access to One-Time Programmable (OTP) memory with layered security architecture. | APB4/AXI4-Lite | 1 |
| UART 16550 | Industry-standard UART with enhanced FIFO capabilities and comprehensive interrupt support. | AXI4-Lite | 4 |
| Log Engine | DMA-based logging system for efficient transfer of log messages to UART interface. | AXI4-Lite | 4 |
| I2C | Full I2C/SMBus/PMBus communication controller supporting both controller and target modes. | AXI4-Lite | 3 |
| I3C | CHIPS Alliance i3c-core with an OCAH AXI4-Lite wrapper. Integration verification remains a work in progress. | AXI4-Lite | 6 |
| AVSBus Controller | Adaptive voltage scaling communication controller for managing voltage rails and power conditions. | APB4/AXI4-Lite | 1 |

| Peripheral | Description | Protocol | Instances |
|---|---|---|---|
| GPIO | General Purpose Input/Output with dual-mode operation and comprehensive interrupt system (61 bonded + 4 unbonded). | AXI4-Lite | 65 |
| Telemetry Receiver | Hardware component for decoding and processing telemetry data from multi-chiplet systems. | AXI4-Lite | 3 |
| Reset Unit | Reset control and synchronization, deals with SMC resets (core, primary), subsystem (chiplet) resets, and the cool reset sequence. | AXI4-Lite | 1 |
| SMC Misc | Wrapper for miscellaneous registers: scratch registers, chip config and NDM reset. | AXI4-Lite | 1 |

### Direct Memory Access (DMA) Engine

The SMC DMA copies data through an AXI master using register-programmed linear or two-dimensional transfers. Software supplies one shared configuration and submits it by reading `NEXT_ID_0`. The engine does not fetch descriptor lists. SMC instantiates one control interface, one stream and one DMA master.

*Table 972. SMC DMA integration*

| Item | SMC configuration |
|---|---|
| Control aperture | `0xC003_8000`–`0xC003_81FF` (512 bytes, SMC-local view). |
| Control path | AXI, 9-bit local address and 64-bit data, 8-bit ID, converted internally to 32-bit register accesses. This is not an AXI4-Lite-only wrapper. |
| Data path | 56-bit addresses, 64-bit data; 3-bit ID at the DMA wrapper and 4-bit ID after the shared DMA/zeroer mux. Backend internal IDs are 2 bits. |
| Access policy | Transfers pass through SMC fabric remapping/filtering; program addresses permitted by that integration. |

### Architecture and Configuration

The frontend captures register state into a request FIFO. The 2D midend expands each command into linear transfers, and the backend performs the AXI reads and writes. The DMA and zeroer share the outgoing fabric path shown below.

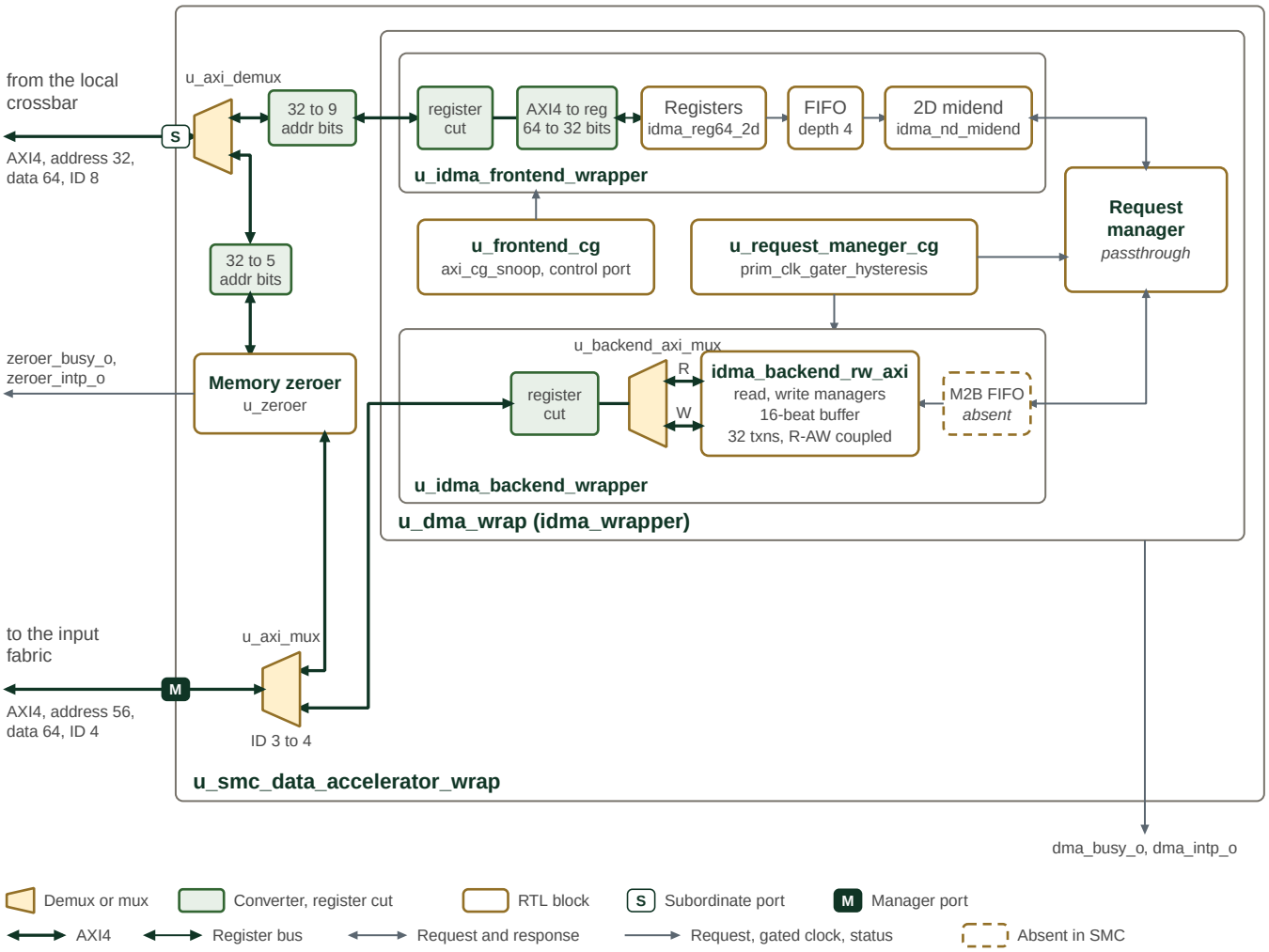


*Figure 62. SMC DMA frontend, backend and shared fabric path*

The DMA and the zeroer share `smc_data_accelerator_wrap`. Its control demux sends the zeroer register window to the zeroer and every other address to the DMA, and each control path narrows the address to its own register window. Its master mux merges the two master ports onto the port to the input fabric and adds one ID bit to tell their responses apart. In the figure, thick lines are AXI4, thin dark lines are the register bus, and grey lines are iDMA requests and responses, gated clocks and status outputs. The dashed box is the manager-to-backend FIFO, which is absent because SMC sets `M2B_FIFO_DEPTH` to 0. With one control interface and one master interface, the request manager is a passthrough, so the arbitration described under [smc-dma-request-arbitration] is not instantiated.

The generic request manager can arbitrate multiple control/master interfaces with round-robin selection, per-master origin tracking and buffered response routing. In the instantiated one-control/one-master SMC configuration it is a passthrough; these generic arbitration features add no extra SMC streams.

*Table 973. SMC DMA elaboration settings*

| Parameter | SMC value | Meaning |
|---|---|---|
| `NUM_CTRL_INTERFACES`, `NUM_CTRL_STREAMS`, `NUM_MST_INTERFACES` | 1 each | One shared configuration, one functional stream and one DMA master. |
| `DMA_MST_MAX_TXNS` | 32 | Backend in-flight AXI limit, from `FabricMaxTrans`. The generic DMA wrapper default is 16. |
| `F2M_FIFO_DEPTH` | 4 | Request FIFO between frontend and 2D midend. |
| `M2B_FIFO_DEPTH` | 0 | Midend-to-backend FIFO bypassed. |

| Parameter | SMC value | Meaning |
|---|---|---|
| `BUFFER_DEPTH` | 16 | Backend realignment buffer depth in beats; the generic wrapper default is 3. It is not a reorder-buffer entry count. |
| `EN_R_AW_COUPLING` | 1 | Backend read/write address coupling enabled. |
| `BYPASS_DMA_CTRL_FLOPS`, `BYPASS_DMA_MST_FLOPS` | 0 each | Interface cuts retained. |
| `CG_HYSTERESIS_W` | 6 | Width of the runtime clock-gating hysteresis input. |

The parameters of `idma_wrapper` are listed in the HTML edition of this manual, under RTL Modules Reference.

The frontend has a bus-aware clock gate; the request manager and backend use a separate activity/hysteresis gate. `cg_enable_i=0` keeps them awake, and test mode overrides gating. Clock gating does not abort transfers or isolate the bus.

### Transfer Configuration

#### Stream and Register Ownership

The generated map contains `NEXT_ID`, `STATUS` and `DONE` banks for 16 streams, but only bank 0 is functional in SMC. Reads of `NEXT_ID_1..15` complete with zero and launch nothing; their `STATUS` and `DONE` registers also read zero. A read of `NEXT_ID_0` is a command, not a status read: any read whose bytes include `NEXT_ID_0`, such as a 64-bit read at `0x48`, launches a transfer; a 32-bit read of the unused word at `0x4C` does not.

Source, destination, length, strides, repetitions and `CONFIG` are registers shared by all software contexts, with no configuration-lock register. The 64-bit fields use low/high 32-bit register halves, which the register path does not update atomically.

*Table 974. DMA register storage and effective field widths*

| Field | CSR storage | Effective width | Use |
|---|---|---|---|
| `SRC_ADDRESS_LO/HI`, `DST_ADDRESS_LO/HI` | 64 bits each | 56 bits each | Byte addresses. |
| `LENGTH_LO/HI` | 64 bits | 56 bits | Bytes per linear copy or per 2D row. Zero-length transfers are rejected. |
| `SRC_STRIDE_LO/HI`, `DST_STRIDE_LO/HI` | 64 bits each | 56 bits each | Address increment between consecutive row starts; source and destination increments are independent. |
| `NUM_REPETITIONS_LO/HI` | 64 bits | 32 bits | Number of rows, including the first. |

The frontend casts register values to these effective widths: upper bits are discarded, not validated as an out-of-range access. Program unused upper bits as zero and ensure all row addresses and lengths fit the permitted address range. Address stepping uses the effective address width.

#### Linear and Two-Dimensional Copies

A linear copy is one repetition with `CONFIG.ENABLED_ND` set; a 2D copy adds source/destination strides and a repetition count. Strides can gather or scatter evenly spaced rows; there is no linked-list walker.

Clearing `CONFIG.ENABLED_ND` forces the midend repetition count to zero. A zero-repetition command produces an internal midend error response without data movement; that response still retires a transfer ID. The control aperture does not expose the error payload, so a retired ID alone is not proof that valid data was copied.

The names above follow the published register map. The RTL frontend calls the configuration `conf`, the ND-enable field `enable_nd`, and the repetition pair `reps_2_low/high`; these refer to the same fields.

The frontend requests incrementing AXI bursts and sets the cache attribute to modifiable. `CONFIG` supplies decoupling, maximum burst-length and length-reduction options; the `CONFIG` register description gives each field's effect. The backend handles alignment and burst splitting; these mechanisms do not grant permission to access a filtered or otherwise invalid address.

#### Submission and Completion

The DMA Engine section of the Programmer's Guide gives the submission and completion sequence.

*Table 975. DMA status and notification*

| Indication | Meaning |
|---|---|
| `STATUS_0[7:0]` | Eight copies of frontend request readiness: `0xFF` means room to accept a command. These are not per-channel backend busy bits. |
| `STATUS_0[8]` / `[9]` | Bit 8 is the midend activity indication; bit 9 is zero. |
| `DONE_0` | Most recently retired transfer ID. No byte-progress or classified software error-status register is provided. |
| DMA interrupt | One-cycle pulse on a falling edge of combined frontend/backend busy, routed to raw CPU vector bit 322 (PLIC ID 323). No local enable, mask or pending register; this is not one interrupt per submitted descriptor. |

Frontend readiness and a zero `STATUS` value do not indicate completion. The transfer-ID generator resets `NEXT_ID` to 2 and `DONE` to 1 and skips zero on wrap. PLIC gateway behavior does not turn successive busy-falling pulses into a counted queue.

### Memory Map and Register Reference

#### Memory Map

**NOTE** The register map below covers all 16 stream banks because the register file is generated at the maximum stream count. With `NUM_CTRL_STREAMS = 1`, only the `_0` bank of `NEXT_ID`, `STATUS`, and `DONE` has any effect; see Stream and Register Ownership.

#### Detailed Register Map

#### iDMA Controller Address Map

Source, destination, length, and control registers for the iDMA data-mover engine.

*Table 976. iDMA Controller Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CONFIG | RW | - |
| 0x4 | STATUS_0 | R | - |
| 0x8 | STATUS_1 | R | - |
| 0xC | STATUS_2 | R | - |
| 0x10 | STATUS_3 | R | - |
| 0x14 | STATUS_4 | R | - |
| 0x18 | STATUS_5 | R | - |
| 0x1C | STATUS_6 | R | - |
| 0x20 | STATUS_7 | R | - |
| 0x24 | STATUS_8 | R | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x28 | STATUS_9 | R | - |
| 0x2C | STATUS_10 | R | - |
| 0x30 | STATUS_11 | R | - |
| 0x34 | STATUS_12 | R | - |
| 0x38 | STATUS_13 | R | - |
| 0x3C | STATUS_14 | R | - |
| 0x40 | STATUS_15 | R | - |
| 0x48 | NEXT_ID_0 | R | - |
| 0x50 | NEXT_ID_1 | R | - |
| 0x58 | NEXT_ID_2 | R | - |
| 0x60 | NEXT_ID_3 | R | - |
| 0x68 | NEXT_ID_4 | R | - |
| 0x70 | NEXT_ID_5 | R | - |
| 0x78 | NEXT_ID_6 | R | - |
| 0x80 | NEXT_ID_7 | R | - |
| 0x88 | NEXT_ID_8 | R | - |
| 0x90 | NEXT_ID_9 | R | - |
| 0x98 | NEXT_ID_10 | R | - |
| 0xA0 | NEXT_ID_11 | R | - |
| 0xA8 | NEXT_ID_12 | R | - |
| 0xB0 | NEXT_ID_13 | R | - |
| 0xB8 | NEXT_ID_14 | R | - |
| 0xC0 | NEXT_ID_15 | R | - |
| 0xC8 | DONE_0 | R | - |
| 0xCC | DONE_1 | R | - |
| 0xD0 | DONE_2 | R | - |
| 0xD4 | DONE_3 | R | - |
| 0xD8 | DONE_4 | R | - |
| 0xDC | DONE_5 | R | - |
| 0xE0 | DONE_6 | R | - |
| 0xE4 | DONE_7 | R | - |
| 0xE8 | DONE_8 | R | - |
| 0xEC | DONE_9 | R | - |
| 0xF0 | DONE_10 | R | - |
| 0xF4 | DONE_11 | R | - |
| 0xF8 | DONE_12 | R | - |
| 0xFC | DONE_13 | R | - |
| 0x100 | DONE_14 | R | - |
| 0x104 | DONE_15 | R | - |
| 0x108 | DST_ADDRESS_LO | RW | - |
| 0x10C | DST_ADDRESS_HI | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x110 | SRC_ADDRESS_LO | RW | - |
| 0x114 | SRC_ADDRESS_HI | RW | - |
| 0x118 | LENGTH_LO | RW | - |
| 0x11C | LENGTH_HI | RW | - |
| 0x120 | DST_STRIDE_LO | RW | - |
| 0x124 | DST_STRIDE_HI | RW | - |
| 0x128 | SRC_STRIDE_LO | RW | - |
| 0x12C | SRC_STRIDE_HI | RW | - |
| 0x130 | NUM_REPETITIONS_LO | RW | - |
| 0x134 | NUM_REPETITIONS_HI | RW | - |

### CONFIG

*Table 977. dma_ctrl.CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10 | `ENABLED_ND` | RW | 0x0 | Repetition enable. 1: the copy is performed NUM_REPETITIONS times, with SRC_STRIDE and DST_STRIDE added to the addresses between rows; set it for every transfer, including a linear copy with NUM_REPETITIONS = 1. 0: the repetition count is forced to zero, so the command completes with an error response, moves no data, and still retires its transfer ID. |
| 9:7 | `DST_MAX_LLEN` | RW | 0x0 | Base-2 logarithm of the maximum write burst length in 64-bit beats, applied when DST_REDUCE_LEN is 1. A write burst does not cross a boundary aligned to 2^(DST_MAX_LLEN+3) bytes; 0 gives single-beat bursts. Ignored when DST_REDUCE_LEN is 0. |
| 6:4 | `SRC_MAX_LLEN` | RW | 0x0 | Base-2 logarithm of the maximum read burst length in 64-bit beats, applied when SRC_REDUCE_LEN is 1. A read burst does not cross a boundary aligned to 2^(SRC_MAX_LLEN+3) bytes; 0 gives single-beat bursts. Ignored when SRC_REDUCE_LEN is 0. |
| 3 | `DST_REDUCE_LEN` | RW | 0x0 | Destination burst-length reduction. 1: write bursts are limited by DST_MAX_LLEN. 0: a write burst is at most 256 beats and does not cross a 2 KiB-aligned boundary. |
| 2 | `SRC_REDUCE_LEN` | RW | 0x0 | Source burst-length reduction. 1: read bursts are limited by SRC_MAX_LLEN. 0: a read burst is at most 256 beats and does not cross a 2 KiB-aligned boundary. |
| 1 | `DECOUPLE_RW` | RW | 0x0 | Read/write decoupling. 0: read and write bursts are formed in lockstep, one write burst per read burst. 1: read and write bursts are formed and issued independently; this mode can deadlock. |
| 0 | `DECOUPLE_AW` | RW | 0x0 | Write-address decoupling. 0: the write address of each burst is held until the first data beat of that burst is ready to be written. 1: write addresses are issued without waiting for write data. |

### STATUS_0

*Table 978. dma_ctrl.STATUS_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 9:0 | BUSY | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_1**

*Table 979. dma_ctrl.STATUS_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9:0 | BUSY | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_2**

*Table 980. dma_ctrl.STATUS_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9:0 | BUSY | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_3**

*Table 981. dma_ctrl.STATUS_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9:0 | BUSY | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_4**

*Table 982. dma_ctrl.STATUS_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9:0 | BUSY | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_5**

*Table 983. dma_ctrl.STATUS_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_6**

*Table 984. dma_ctrl.STATUS_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_7**

*Table 985. dma_ctrl.STATUS_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_8**

*Table 986. dma_ctrl.STATUS_8 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_9**

*Table 987. dma_ctrl.STATUS_9 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_10**

*Table 988. dma_ctrl.STATUS_10 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_11**

*Table 989. dma_ctrl.STATUS_11 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_12**

*Table 990. dma_ctrl.STATUS_12 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_13**

*Table 991. dma_ctrl.STATUS_13 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_14**

*Table 992. dma_ctrl.STATUS_14 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

**STATUS_15**

*Table 993. dma_ctrl.STATUS_15 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9:0 | `BUSY` | R | 0x0 | Status for the active stream: bits [7:0] each indicate readiness to accept another command, bit 8 indicates midend activity, and bit 9 is zero. These bits do not report backend busy or prove transfer completion; use DONE for completed transfer IDs. Unused streams read zero. |

### NEXT_ID_0

*Table 994. dma_ctrl.NEXT_ID_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_1

*Table 995. dma_ctrl.NEXT_ID_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_2

*Table 996. dma_ctrl.NEXT_ID_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_3

*Table 997. dma_ctrl.NEXT_ID_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_4

*Table 998. dma_ctrl.NEXT_ID_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_5

*Table 999. dma_ctrl.NEXT_ID_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_6

*Table 1000. dma_ctrl.NEXT_ID_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_7

*Table 1001. dma_ctrl.NEXT_ID_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_8

*Table 1002. dma_ctrl.NEXT_ID_8 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_9

*Table 1003. dma_ctrl.NEXT_ID_9 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_10

*Table 1004. dma_ctrl.NEXT_ID_10 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_11

*Table 1005. dma_ctrl.NEXT_ID_11 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_12

*Table 1006. dma_ctrl.NEXT_ID_12 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_13

*Table 1007. dma_ctrl.NEXT_ID_13 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_14

*Table 1008. dma_ctrl.NEXT_ID_14 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### NEXT_ID_15

*Table 1009. dma_ctrl.NEXT_ID_15 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Reading this register starts the DMA transfer. Returns an ID value for the cumulative number of transfers. Returns 0 if command was not set up correctly |

### DONE_0

*Table 1010. dma_ctrl.DONE_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_1

*Table 1011. dma_ctrl.DONE_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_2

*Table 1012. dma_ctrl.DONE_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_3

*Table 1013. dma_ctrl.DONE_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_4

*Table 1014. dma_ctrl.DONE_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_5

*Table 1015. dma_ctrl.DONE_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_6

*Table 1016. dma_ctrl.DONE_6 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_7

*Table 1017. dma_ctrl.DONE_7 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_8

*Table 1018. dma_ctrl.DONE_8 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_9

*Table 1019. dma_ctrl.DONE_9 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_10

*Table 1020. dma_ctrl.DONE_10 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_11

*Table 1021. dma_ctrl.DONE_11 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_12

*Table 1022. dma_ctrl.DONE_12 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_13

*Table 1023. dma_ctrl.DONE_13 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_14

*Table 1024. dma_ctrl.DONE_14 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DONE_15

*Table 1025. dma_ctrl.DONE_15 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | Holds the cumulative number of completed transfers. Only accumulates when response is seen |

### DST_ADDRESS_LO

*Table 1026. dma_ctrl.DST_ADDRESS_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Destination address for DMA to write to, lower 32 bits |

### DST_ADDRESS_HI

*Table 1027. dma_ctrl.DST_ADDRESS_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Destination address for DMA to write to, upper 32 bits. Bits [31:24] (combined address bits [63:56]) are discarded, not checked; program them as zero. |

### SRC_ADDRESS_LO

*Table 1028. dma_ctrl.SRC_ADDRESS_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Source address for DMA to read from, lower 32 bits. |

### SRC_ADDRESS_HI

*Table 1029. dma_ctrl.SRC_ADDRESS_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Source address for DMA to read from, upper 32 bits. Bits [31:24] (combined address bits [63:56]) are discarded, not checked; program them as zero. |

### LENGTH_LO

*Table 1030. dma_ctrl.LENGTH_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Number of bytes to move, lower 32 bits. |

### LENGTH_HI

*Table 1031. dma_ctrl.LENGTH_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Number of bytes to move, upper 32 bits. Bits [31:24] (combined length bits [63:56]) are discarded, not checked; program them as zero. |

#### DST_STRIDE_LO

*Table 1032. dma_ctrl.DST_STRIDE_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Stride amount on destination address side, lower 32 bits |

#### DST_STRIDE_HI

*Table 1033. dma_ctrl.DST_STRIDE_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Stride amount on destination address side, upper 32 bits. Bits [31:24] (combined stride bits [63:56]) are discarded, not checked; program them as zero. |

#### SRC_STRIDE_LO

*Table 1034. dma_ctrl.SRC_STRIDE_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Stride amount on source address side, lower 32 bits |

#### SRC_STRIDE_HI

*Table 1035. dma_ctrl.SRC_STRIDE_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Stride amount on source address side, upper 32 bits. Bits [31:24] (combined stride bits [63:56]) are discarded, not checked; program them as zero. |

#### NUM_REPETITIONS_LO

*Table 1036. dma_ctrl.NUM_REPETITIONS_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Number of rows to copy, including the first, lower 32 bits. SRC_STRIDE and DST_STRIDE are added to the addresses between rows. Takes effect only when CONFIG.ENABLED_ND is 1. |

#### NUM_REPETITIONS_HI

*Table 1037. dma_ctrl.NUM_REPETITIONS_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | Number of rows to copy, upper 32 bits. The DMA uses only the low 32 bits of the row count, so this register is discarded, not checked; program it as zero. |

### Zeroer

The SMC Memory Zeroer efficiently initializes large memory regions to zero without CPU intervention. The unit uses a state machine-based architecture in a single clock domain, with separately gated clocks for the clear engine and the register interface.

Table 1038 summarises the zeroer control aperture and fabric integration.

*Table 1038. SMC Memory Zeroer Integration*

| Parameter | Description |
|---|---|
| Address Range | 0xC003_8200 - 0xC003_83FF (512 bytes) |
| Control Interface | AXI4 (5-bit address, 64-bit data, 8-bit ID), converted internally to AXI4-Lite for the registers |
| Data Interface | AXI4 Master (56-bit address, 64-bit data, 3-bit ID; 4-bit ID after the shared DMA/zeroer mux) |
| Maximum Burst Length | 256 beats (`AWLEN = 255`) |
| Outstanding Transactions | Counted by a 32-bit counter; no 32-transaction issuance limit or counter-based back-pressure |

### Use Cases

- **System Initialization**: Clear memory regions during boot or reset sequences
- **Security Operations**: Sanitize sensitive data areas
- **Memory Management**: Initialize allocated memory blocks
- **Error Recovery**: Clear corrupted memory regions

### Architecture

The DMA and zeroer share the data-accelerator fabric path shown in SMC DMA frontend, backend and shared fabric path.

### State Machine

The zeroer state machine has three operating states and an error state (Table 1039).

*Table 1039. SMC zeroer state machine*

| State | Description |
|---|---|
| ST_IDLE (3'b001) | Idle state waiting for configuration and trigger |
| ST_ISSUE_ADDR (3'b010) | Address phase - issuing AXI write addresses |
| ST_ISSUE_DATA (3'b100) | Data phase - streaming zero data for all beats |
| ST_ERROR (3'b000) | Entered from any illegal state encoding; holds the zeroer busy, with no interrupt, until reset |

Figure 63 shows the state machine and its AXI write flow.

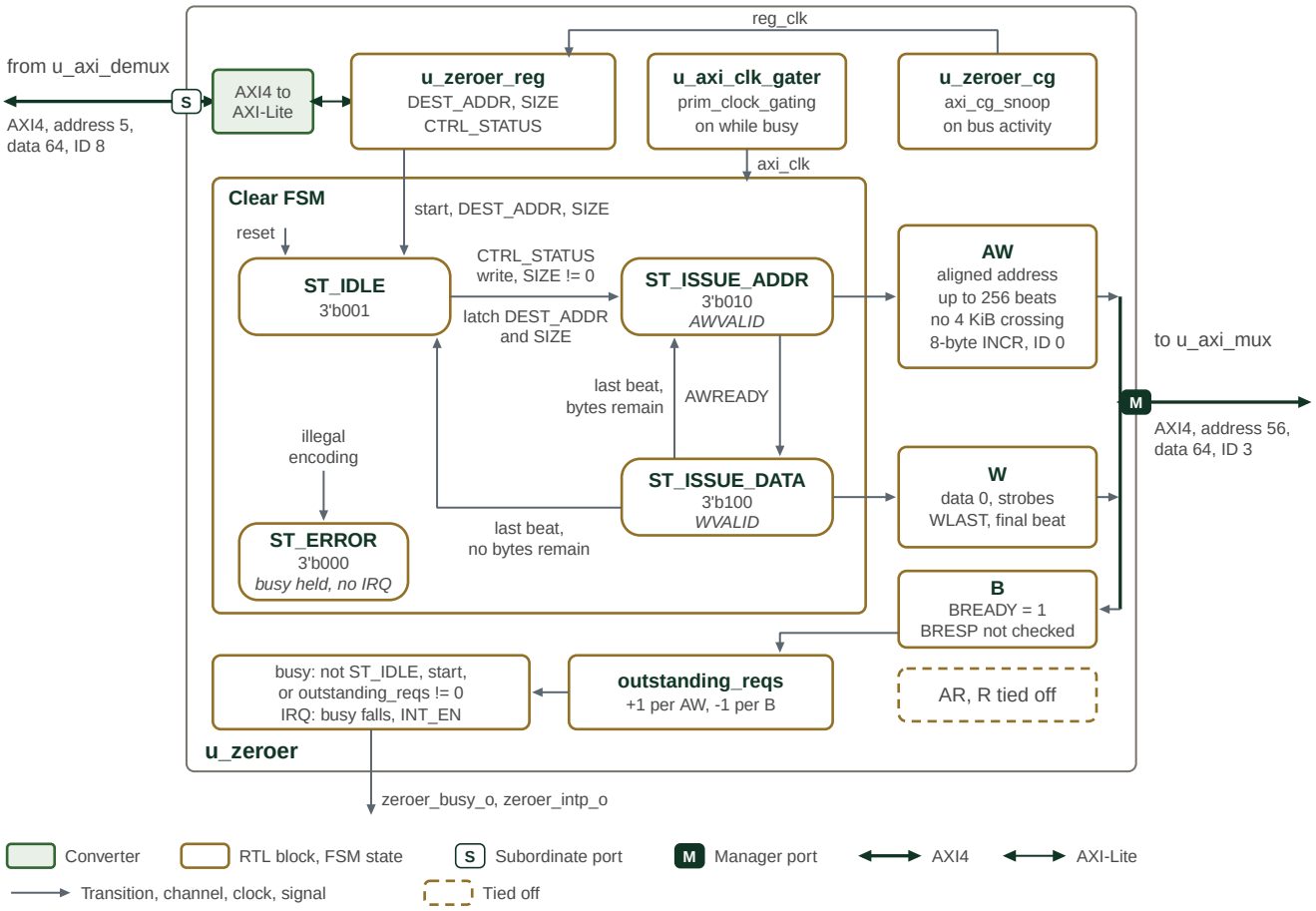


*Figure 63. SMC memory zeroer state machine and AXI write flow*

Rounded boxes are the states of the clear state machine, with their encodings; grey arrows are transitions, labelled with their conditions. A write to `CTRL_STATUS` while `SIZE` is non-zero leaves `ST_IDLE`. Each pass through `ST_ISSUE_ADDR` and `ST_ISSUE_DATA` issues one write burst of zeros, and the machine returns to `ST_IDLE` once no bytes remain. `ST_ERROR` is entered only from an illegal state encoding and is left only by reset. The write-response channel is always ready and its response code is not checked; the outstanding-request count decides when the zeroer stops being busy.

### Operation Flow

1. **Configuration**: Software writes destination address and size to registers
2. **Trigger**: Write to control/status register initiates operation
3. **Address Phase**: Hardware calculates optimal burst patterns and issues write addresses
4. **Data Phase**: Hardware streams zero data matching issued address beats
5. **Completion**: State machine returns to idle with status update and optional interrupt

### AXI Master Interface

Table 1040 summarises the zeroer's AXI master interface.

*Table 1040. SMC zeroer AXI4 Master Characteristics*

| Parameter | Specification |
|---|---|
| Address Width | 56-bit addressing for large system integration |

| Parameter | Specification |
|---|---|
| Data Width | 64-bit data path |
| ID Width | 4-bit transaction ID (internal 3-bit usage) |
| User Signals | 12-bit user field for system-specific attributes |
| Maximum Burst Length | 256 beats (`AWLEN = 255`) |
| Burst Calculation | Automatic optimal burst sizing |
| Address Alignment | Intelligent handling of unaligned start addresses |
| Boundary Handling | Automatic burst fragmentation at page boundaries |

### Outstanding Transaction Management

The zeroer counts accepted write addresses and consumed write responses in a 32-bit outstanding-request counter. The counter determines whether responses remain pending; it does not impose a limit of 32 transactions or throttle address issuance. AXI `AWREADY` and `WREADY` provide downstream back-pressure.

The response channel is consumed without checking `BRESP`. Completion means that all responses have arrived, not that every write succeeded. The internal error state handles an illegal state-machine encoding, not AXI bus errors. The Memory Zeroer procedure in the Programmer's Guide covers validating a request and verifying a clear.

### Clock Gating

The zeroer has one input clock, `clk_i`, and gates two branches of it: the clear state machine and AXI master clock (`axi_clk`), and the register interface clock (`reg_clk`). Both are in the `clk_i` domain. Table 1041 gives the condition under which each branch runs.

*Table 1041. SMC zeroer Clock Gating Control*

| Branch | Runs while |
|---|---|
| Clear state machine and AXI master (`axi_clk`) | `cg_enable_i` is low, the zeroer is busy (`zeroer_busy_o`), or `rst_ni` is asserted. A `prim_clock_gating` cell gates it. |
| Register interface (`reg_clk`) | `cg_enable_i` is low, or the control bus has a transaction in flight and for `cg_hysteresis_i` idle cycles after it. A bus-aware gate (`axi_cg_snoop`) gates it. |
| Test Support | Test mode overrides both gates. |

### Control and Status

### Operation Control

Table 1042 defines the registers used to configure and start a zeroing operation.

*Table 1042. SMC zeroer Configuration and Control*

| Register | Description |
|---|---|
| DEST_ADDR | 64-bit byte address for destination memory region |
| SIZE | 64-bit byte count for zeroing operation |
| CTRL_STATUS | Control and status register (trigger, busy status) |
| CTRL_STATUS.INT_EN | Enables the completion pulse |

#### Status Features

- **Busy Status**: Real-time indication of zeroing operation in progress
- **Completion Status**: Updates when all write responses received
- **Error Reporting**: No software-visible AXI error report; failed write responses are not distinguished from successful responses
- **Interrupt Generation**: Optional interrupt on operation completion

#### Operation Details

Table 1043 summarises zero-data generation and transfer control.

*Table 1043. SMC zeroer Data Generation and Control*

| Feature | Description |
|---|---|
| Zero Data Generation | Hardware generates zero data for all beats |
| Strobe Calculation | Correct byte strobes for partial transfers and unaligned addresses |
| Last Beat Handling | Proper w_last assertion for burst completion |
| Size Tracking | Accurate remaining size calculation throughout operation |
| Overflow Protection | Overflow assertions are simulation checks; there is no hardware overflow-rejection status or recovery control |
| Abort | No software abort control or guarantee that a reset safely drains outstanding writes |

### Memory Map and Register Reference

#### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

#### Detailed Register Map

#### Zeroer Control Address Map

Control and status registers for the AXI memory-zeroing engine.

*Table 1044. Zeroer Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | DEST_ADDR | RW | - |
| 0x8 | SIZE | RW | - |
| 0x10 | CTRL_STATUS | RW | Writing this register will trigger start of zeroer. The write is ignored while SIZE is 0. |

#### DEST_ADDR

*Table 1045. zeroer_ctrl.DEST_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | DEST_ADDR | RW | 0x0 | Byte address to write zeros to |

#### SIZE

*Table 1046. zeroer_ctrl.SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `SIZE` | RW | 0x0 | Size in bytes of zeros to write |

#### CTRL_STATUS

*Table 1047. zeroer_ctrl.CTRL_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:33 | `Reserved` | - | - | Reserved |
| 32 | `STATUS` | R | 0x0 | Indicates whether the zeroer is busy. 1 means the zeroer is active, has pending work, or is in an error state; 0 means it is idle with no pending or outstanding work, including after completion. |
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `INT_EN` | RW | 0x0 | If enabled, an interrupt will be outputted once zeroer completes. |

### Watchdog Timers

#### Operation

Each CPU core includes a dedicated watchdog timer (at 0xC000_0000 + N * 0x400), providing per-core timeout monitoring. Reset effects are described in Clocks and Resets. Base addresses and interrupt-related context for the WDT are summarized in Interrupts.

Timeouts are configurable to suit different workloads, and staged warnings (followed by reset) allow software intervention before hardware reset is triggered.

Watchdog monitoring can be disabled during debugging to prevent false triggers, supporting uninterrupted development and troubleshooting.

The watchdog interrupt is `CTRL.wdogip0`. Hardware sets it on every cycle the scaled count is at or above `CMP`, so it cannot be cleared while that holds. Feeding the watchdog resets the count but leaves `wdogip0` set. The watchdog interrupt clear sequence is in the Programmer's Guide.

#### Register Access

Each watchdog is locked out of reset, and a write to any register other than KEY locks it again. Software changes one register at a time:

1. Write `0x51F15E` to KEY. KEY reads 1 while the watchdog is unlocked.
2. Write the target register. The watchdog locks again on this write, so the next write needs a new KEY write first.

A write acts only when its byte enables cover the register's width, and a narrower write completes with an OKAY response without changing anything, so software cannot detect a dropped write from the response. Software uses 32-bit stores for every register except CTRL, which accepts byte writes. The Detailed Register Map below gives the width each register requires.

### Memory Map and Register Reference

### Memory Map

Register offsets are relative to the block base in the subsystem memory map. The detailed map below defines each register and its fields.

### Detailed Register Map

### Watchdog Timer Address Map

Watchdog timer configuration, counter, compare, and feed and key registers.

A write takes effect only if its byte enables cover all four bytes of COUNT, COUNT_HI, FEED or KEY, or bytes 0 and 1 of SCALED_COUNT or CMP; each CTRL byte is written on its own. Any other write returns OKAY and changes nothing, not even the lock.

Each effective write to a register other than KEY locks the watchdog, and changes no field if it was already locked. A 64-bit write at 0x18 writes FEED and KEY together and leaves the watchdog locked.

*Table 1048. Watchdog Timer Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CTRL | RW | KEY reg must be written to before any other register in this addrmap can be written to, unlocking the WDT. After writing to another register, the WDT will be locked again, and KEY must be written to again to unlock it. |
| 0x8 | COUNT | RW | - |
| 0xC | COUNT_HI | W | - |
| 0x10 | SCALED_COUNT | RW | - |
| 0x18 | FEED | RW | - |
| 0x1C | KEY | RW | - |
| 0x20 | CMP | RW | - |

### CTRL

*Table 1049. wdt.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `rsvd3` | R | 0x0 | Reserved |
| 28 | `wdogip0` | RW | 0x0 | Pending interrupt. Hardware sets this bit on every cycle the scaled count has reached the compare value. To clear it, write KEY and then write 0 to this bit while the scaled count is below the compare value. Writing FEED does not clear it. |
| 27:14 | `rsvd2` | R | 0x0 | Reserved |
| 13 | `wdogcoreawake` | RW | 0x0 | Enable WDT to only count when the core is not in reset |
| 12 | `wdogenalways` | RW | 0x0 | Enable WDT to always count. This has priority over wdogcoreawake |
| 11:10 | `rsvd1` | R | 0x0 | Reserved |
| 9 | `wdogzerocmp` | RW | 0x0 | Enable cycle counter to be reset to 0 when compare value has been reached |
| 8 | `wdogrsten` | RW | 0x0 | Enables a sticky reset signal to outputted to top level, can be used for a second stage wdt |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:4 | `rsvd0` | R | 0x0 | Reserved |
| 3:0 | `wdogscale` | RW | 0x0 | WDT counter scale value. Scales the current cycle count (time) by count / 2 ^ wdogscale value |

### COUNT

*Table 1050. wdt.COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30:0 | `wdogcount` | RW | 0x0 | 31 bit wdt counter registers. On write, the counter will be set to the written value, then continue counting up. KEY reg must be written to before any other register in this addrmap can be written to, unlocking the WDT. After writing to another register, the WDT will be locked again, and KEY must be written to again to unlock it. |

### COUNT_HI

*Table 1051. wdt.COUNT_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `wdogcounthi` | W | 0x0 | Upper word of the counter. The counter is 31 bits wide and lies entirely in COUNT, so this register has no storage and reads 0. A write covering all four bytes leaves the counter unchanged but locks the WDT, so KEY must be written again before the next write. |

### SCALED_COUNT

*Table 1052. wdt.SCALED_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `wdogs` | RW | 0x0 | Scaled value of wdt counter. (i.e. count >> wdogscale). Writing to this reg has no effect, but will lock the WDT (meaning KEY must be written to KEY reg to unlock it again). KEY reg must be written to before any other register in this addrmap can be written to, unlocking the WDT. After writing to another register, the WDT will be locked again, and KEY must be written to again to unlock it. |

### FEED

*Table 1053. wdt.FEED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `wdogfeed` | RW | 0x0 | Feed register. Writing 0xD09F00D will reset the counter value back to 0. Feeding does not clear CTRL.wdogip0. KEY reg must be written to before any other register in this addrmap can be written to, unlocking the WDT. After writing to another register, the WDT will be locked again, and KEY must be written to again to unlock it. |

### KEY

*Table 1054. wdt.KEY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `wdogkey` | RW | 0x0 | Key register. Magic key (0x51F15E) must be written to this register before a write to any other register in this addrmap. Must be written every time. Reading this reg returns 1 if key has been written to it and it is unlocked, 0 otherwise. |

#### CMP

*Table 1055. wdt.CMP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `wdogcmp0` | RW | 0x1000 | Compare register. Value that gets compared to wdogs. KEY reg must be written to before any other register in this addrmap can be written to, unlocking the WDT. After writing to another register, the WDT will be locked again, and KEY must be written to again to unlock it. |

### Mailbox Unit

#### Overview

The Mailbox Unit connects two software endpoints through a pair of FIFOs. Each endpoint has a register aperture: writing `WRITE_DATA` sends one word to the peer, and reading `READ_DATA` consumes one word sent by the peer. A single AXI4-Lite slave port serves all pairs.

`NUM_MAILBOXES` counts bidirectional pairs, each with two directional apertures and two interrupt outputs. The SMC instance provides 32 pairs (64 apertures); the SEP instance provides 8 pairs (16 apertures). Aperture names describe intended use; both ends remain reachable through the same slave port. Access isolation is the subsystem's responsibility, not a property conferred by naming an aperture inbound or outbound.

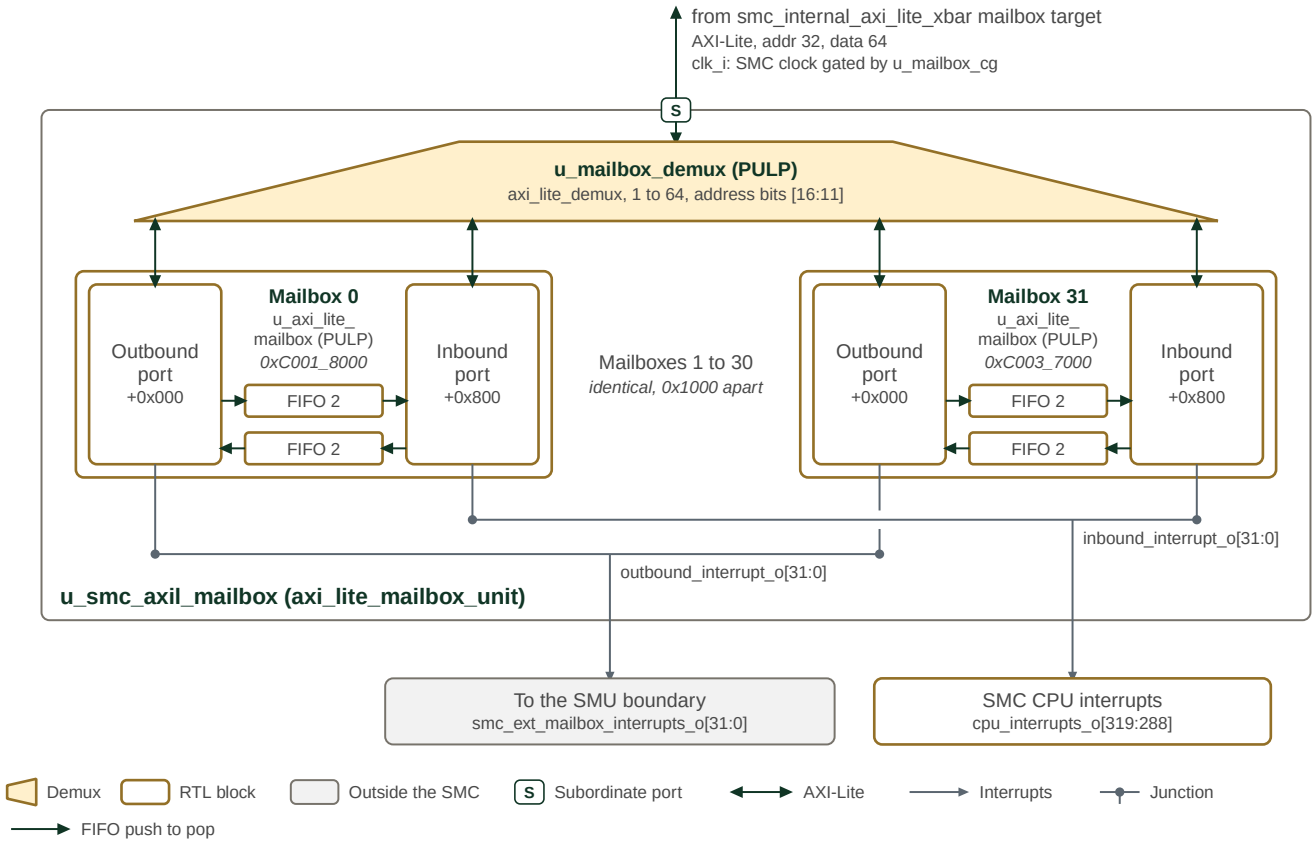


*Figure 64. SMC mailbox wrapper and interrupt routing*

Figure 64 shows the SMC instance of the mailbox, `u_smc_axil_mailbox`, and its interrupt connections. The SMC configures 32 mailboxes of depth 2, with 32-bit address and 64-bit data. The port comes from the mailbox target of `smc_internal_axi_lite_xbar`, and the unit runs on a copy of the SMC clock that `u_mailbox_cg` gates while the port is idle. The demux `u_mailbox_demux`, a PULP `axi_lite_demux`, selects one of 64 ports from address bits [16:11] above `0xC001_8000`. Each mailbox, `gen_mailbox[m].u_axi_lite_mailbox`, is a PULP `axi_lite_mailbox`; the figure puts the source project in parentheses. Mailbox `m` places its outbound port at `0xC001_8000` + `m` × `0x1000` and its inbound port `0x800` above it.

Within a mailbox, the arrows between the ports show the direction of data: `WRITE_DATA` in one port pushes the FIFO that `READ_DATA` in the other port pops. The 32 inbound-port interrupts are SMC CPU interrupts 288 to 319. The 32 outbound-port interrupts leave the SMC on `smc_ext_mailbox_interrupts_o`, which the SMU passes out of its boundary.

The SEP holds a second instance of this unit with eight mailboxes, described in the SEP fabric chapter. The outbound interrupts of that instance reach the SMC as peripheral interrupts 0 to 7, SMC CPU interrupts 256 to 263.

### Architecture

The parameters and ports of `axi_lite_mailbox_unit` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Paired FIFOs

An `axi_lite_demux` selects one of `2 * NUM_MAILBOXES` register apertures. Within pair N, the outbound aperture writes FIFO A and reads FIFO B; the inbound aperture writes FIFO B and reads FIFO A. The FIFOs are internal to the unit. Flushing an aperture's write FIFO therefore discards data its peer would read; flushing its read FIFO discards data the peer has already sent.

WRITE_DATA enqueues one word when space is available. Inactive AXI byte lanes are written as zero; a partial write does not merge with an existing FIFO word. READ_DATA returns and pops the oldest received word. Software must use status to avoid full writes and empty reads; the interface reports errors rather than waiting for FIFO space or data. The error responses are described below.

### Interrupts

Each aperture has its own WIRQT, RIRQT, IRQS, IRQEN and IRQP. inbound_interrupt_o[N] and outbound_interrupt_o[N] are active-high level outputs. Each is the OR of IRQP = IRQS & IRQEN for its aperture.

*Table 1056. Mailbox interrupt sources*

| IRQS bit | Source | Set condition |
|---|---|---|
| 0 | Write threshold | Occupancy of the aperture's write FIFO is strictly greater than WIRQT. This indicates queued outbound data, not successful peer receipt. |
| 1 | Read threshold | Occupancy of its read FIFO is strictly greater than RIRQT. |
| 2 | Access error | A full-FIFO write or empty-FIFO read. ERROR_FLAGS distinguishes which access failed. |

These status bits latch independently of IRQEN. Acknowledge selected bits by writing ones to IRQS; reading it does not clear them. Threshold status can reassert if the occupancy condition remains true. WIRQT and RIRQT writes are clamped to MAILBOX_DEPTH - 1, so a threshold of zero detects any queued word and the maximum threshold detects a full FIFO.

STATUS exposes live FIFO-empty/full and threshold conditions. It is distinct from the latched IRQS. Reading ERROR_FLAGS clears the detailed error flags, but does not acknowledge the shared IRQS error bit.

### Clock and Reset

The unit and its FIFOs use clk_i; there is no internal clock-domain crossing. rst_ni is asynchronous and active low. Reset empties the FIFOs and clears thresholds, error flags, interrupt status and enables. test_en_i supplies the AXI demultiplexer's test input. Follow subsystem reset and access-isolation requirements when either endpoint is active.

### Interface and Programming

### External Interface

A single AXI4-Lite slave (mailbox_axi_req_i / mailbox_axi_resp_o) serves all apertures; its address/data widths and channel/aggregate struct types are elaboration parameters. The inbound_interrupt_o and outbound_interrupt_o vectors each carry NUM_MAILBOXES active-high level interrupts, one per aperture.

The full parameter and port list of axi_lite_mailbox_unit is in the HTML edition of this manual, under RTL Modules Reference.

The per-aperture B and R responses pass through spill registers. Request arbitration, arrival of both write channels, and response backpressure also affect observed latency; software must not assume a fixed completion cycle. Reads and writes use separate AXI4-Lite channels but operate on the same FIFO and status state.

### Access Results

*Table 1057. Mailbox access results*

| Access | Result |
|---|---|
| Read READ_DATA while empty | SLVERR, data 0xFEEDDEAD; sets the read-error flag and IRQS[2]. |

| Access | Result |
|---|---|
| Write `WRITE_DATA` while full | `SLVERR`; data discarded; sets the write-error flag and `IRQS[2]`. |
| Read write-only `WRITE_DATA` | `OKAY`, data `0xFEEDC0DE`; no FIFO access or error flag. |
| Write `READ_DATA`, `STATUS`, `ERROR_FLAGS` or `IRQP` | Ignored with `SLVERR`. |
| Access to an unmapped offset inside an aperture (`0x50` and above) | Stated in the SMC and SEP memory maps |
| Read `ERROR_FLAGS` | Returns and clears both detailed error flags. A new error in the same cycle is retained. |
| Write `CTRL` flush bits | Pulses the selected FIFO flush; the bits are not stored configuration and read back as zero. |

### Sending and Receiving

The register interface does not allocate FIFO ownership to clients, and status and data accesses are not one atomic operation. The Programmer's Guide gives the mailbox usage rules.

### Memory Map and Register Reference

### Memory Map

With the default `MAILBOX_SIZE=0x800`, each directional aperture occupies 2 KiB and each pair occupies 4 KiB. The SMC 32-pair map spans `0x20000` bytes, and the SEP 8-pair map spans `0x8000` bytes; the generic unit's span is `2 * MAILBOX_SIZE * NUM_MAILBOXES`. The SMC and SEP wrapper layouts are generated from their SystemRDL compositions:

*Table 1058. SMC Mailbox Aperture Layout*

| Address Range | Size | Decoded Extent | Unit | Instances | Stride |
|---|---|---|---|---|---|
| BASE + 0x0 – BASE + 0x7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x800 – BASE + 0xFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1000 – BASE + 0x17FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1800 – BASE + 0x1FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x2000 – BASE + 0x27FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x2800 – BASE + 0x2FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x3000 – BASE + 0x37FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x3800 – BASE + 0x3FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x4000 – BASE + 0x47FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x4800 – BASE + 0x4FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x5000 – BASE + 0x57FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x5800 – BASE + 0x5FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x6000 – BASE + 0x67FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride |
|---|---|---|---|---|---|
| BASE + 0x6800 – BASE + 0x6FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x7000 – BASE + 0x77FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x7800 – BASE + 0x7FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x8000 – BASE + 0x87FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x8800 – BASE + 0x8FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x9000 – BASE + 0x97FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x9800 – BASE + 0x9FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xA000 – BASE + 0xA7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xA800 – BASE + 0xAFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xB000 – BASE + 0xB7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xB800 – BASE + 0xBFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xC000 – BASE + 0xC7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xC800 – BASE + 0xCFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xD000 – BASE + 0xD7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xD800 – BASE + 0xDFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xE000 – BASE + 0xE7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xE800 – BASE + 0xEFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xF000 – BASE + 0xF7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0xF800 – BASE + 0xFFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x10000 – BASE + 0x107FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x10800 – BASE + 0x10FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x11000 – BASE + 0x117FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x11800 – BASE + 0x11FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x12000 – BASE + 0x127FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x12800 – BASE + 0x12FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride |
|---|---|---|---|---|---|
| BASE + 0x13000 – BASE + 0x137FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x13800 – BASE + 0x13FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x14000 – BASE + 0x147FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x14800 – BASE + 0x14FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x15000 – BASE + 0x157FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x15800 – BASE + 0x15FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x16000 – BASE + 0x167FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x16800 – BASE + 0x16FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x17000 – BASE + 0x177FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x17800 – BASE + 0x17FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x18000 – BASE + 0x187FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x18800 – BASE + 0x18FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x19000 – BASE + 0x197FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x19800 – BASE + 0x19FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1A000 – BASE + 0x1A7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1A800 – BASE + 0x1AFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1B000 – BASE + 0x1B7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1B800 – BASE + 0x1BFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1C000 – BASE + 0x1C7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1C800 – BASE + 0x1CFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1D000 – BASE + 0x1D7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1D800 – BASE + 0x1DFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1E000 – BASE + 0x1E7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1E800 – BASE + 0x1EFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1F000 – BASE + 0x1F7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride |
|---|---|---|---|---|---|
| BASE + 0x1F800 – BASE + 0x1FFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |

*Table 1059. SEP Mailbox Aperture Layout*

| Address Range | Size | Decoded Extent | Unit | Instances | Stride |
|---|---|---|---|---|---|
| BASE + 0x0 – BASE + 0x7FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x800 – BASE + 0xFFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1000 – BASE + 0x17FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x1800 – BASE + 0x1FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x2000 – BASE + 0x27FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x2800 – BASE + 0x2FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x3000 – BASE + 0x37FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x3800 – BASE + 0x3FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x4000 – BASE + 0x47FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x4800 – BASE + 0x4FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x5000 – BASE + 0x57FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x5800 – BASE + 0x5FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x6000 – BASE + 0x67FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x6800 – BASE + 0x6FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x7000 – BASE + 0x77FF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |
| BASE + 0x7800 – BASE + 0x7FFF | 2 KiB | 80 B | AXI-Lite Mailbox Address Map | 1 instance | - |

### Per-Mailbox Register Map

Each aperture contains ten registers. Offsets are spaced by `DATA_WIDTH`/8, so they are 8 bytes apart in the 64-bit configurations used today. The generated detailed register map below is authoritative for offsets, widths, access, and field descriptions. The `CTRL` flush bits are pulses rather than stored state, so reads return zero with an OKAY response.

The SMC and SEP memory maps state how an offset above the registers answers.

### Address Examples

For mailbox pair `N`, the outbound aperture begins at `BASE + N * 0x1000`; the inbound aperture begins 0x800 bytes later. A register address is the aperture base plus the generated register offset. For example, in pair zero, `READ_DATA` is at `BASE +`

`0x8` outbound and `BASE + 0x808` inbound.

### Detailed Register Map

### AXI-Lite Mailbox Address Map

Register interface for a single AXI-Lite mailbox: data FIFOs, level and status flags, and interrupt controls.

*Table 1060. AXI-Lite Mailbox Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | WRITE_DATA | W | - |
| 0x8 | READ_DATA | R | - |
| 0x10 | STATUS | R | - |
| 0x18 | ERROR_FLAGS | R | - |
| 0x20 | WIRQT | RW | - |
| 0x28 | RIRQT | RW | - |
| 0x30 | IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x38 | IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x40 | IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x48 | CTRL | W | - |

#### WRITE_DATA

*Table 1061. axil_mailbox.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

#### READ_DATA

*Table 1062. axil_mailbox.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

#### STATUS

*Table 1063. axil_mailbox.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### ERROR_FLAGS

*Table 1064. axil_mailbox.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### WIRQT

*Table 1065. axil_mailbox.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### RIRQT

*Table 1066. axil_mailbox.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### IRQS

*Table 1067. axil_mailbox.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### IRQEN

*Table 1068. axil_mailbox.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### IRQP

*Table 1069. axil_mailbox.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### CTRL

*Table 1070. axil_mailbox.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### Subsystem Wrapper Maps

The wrapper maps describe the generated SMC and SEP mailbox compositions. Their offsets are relative to the corresponding wrapper aperture.

### SMC AXI-Lite Mailbox Wrapper Address Map

SMC-side wrapper instantiating the AXI-Lite mailbox pairs. Each outbound_mailbox_N / inbound_mailbox_N pair is the two apertures of one bidirectional mailbox and shares a single pair of FIFOs: data written through WRITE_DATA in one aperture is what READ_DATA in the other aperture returns. The outbound/inbound naming reflects intended use, not an access restriction.

*Table 1071. SMC AXI-Lite Mailbox Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | outbound_mailbox_0.WRITE_DATA | W | - |
| 0x8 | outbound_mailbox_0.READ_DATA | R | - |
| 0x10 | outbound_mailbox_0.STATUS | R | - |
| 0x18 | outbound_mailbox_0.ERROR_FLAGS | R | - |
| 0x20 | outbound_mailbox_0.WIRQT | RW | - |
| 0x28 | outbound_mailbox_0.RIRQT | RW | - |
| 0x30 | outbound_mailbox_0.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x38 | outbound_mailbox_0.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x40 | outbound_mailbox_0.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x48 | outbound_mailbox_0.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x800 | inbound_mailbox_0.WRITE_DATA | W | - |
| 0x808 | inbound_mailbox_0.READ_DATA | R | - |
| 0x810 | inbound_mailbox_0.STATUS | R | - |
| 0x818 | inbound_mailbox_0.ERROR_FLAGS | R | - |
| 0x820 | inbound_mailbox_0.WIRQT | RW | - |
| 0x828 | inbound_mailbox_0.RIRQT | RW | - |
| 0x830 | inbound_mailbox_0.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x838 | inbound_mailbox_0.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x840 | inbound_mailbox_0.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x848 | inbound_mailbox_0.CTRL | W | - |
| 0x1000 | outbound_mailbox_1.WRITE_DATA | W | - |
| 0x1008 | outbound_mailbox_1.READ_DATA | R | - |
| 0x1010 | outbound_mailbox_1.STATUS | R | - |
| 0x1018 | outbound_mailbox_1.ERROR_FLAGS | R | - |
| 0x1020 | outbound_mailbox_1.WIRQT | RW | - |
| 0x1028 | outbound_mailbox_1.RIRQT | RW | - |
| 0x1030 | outbound_mailbox_1.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1038 | outbound_mailbox_1.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1040 | outbound_mailbox_1.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1048 | outbound_mailbox_1.CTRL | W | - |
| 0x1800 | inbound_mailbox_1.WRITE_DATA | W | - |
| 0x1808 | inbound_mailbox_1.READ_DATA | R | - |
| 0x1810 | inbound_mailbox_1.STATUS | R | - |
| 0x1818 | inbound_mailbox_1.ERROR_FLAGS | R | - |
| 0x1820 | inbound_mailbox_1.WIRQT | RW | - |
| 0x1828 | inbound_mailbox_1.RIRQT | RW | - |
| 0x1830 | inbound_mailbox_1.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1838 | inbound_mailbox_1.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1840 | inbound_mailbox_1.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1848 | inbound_mailbox_1.CTRL | W | - |
| 0x2000 | outbound_mailbox_2.WRITE_DATA | W | - |
| 0x2008 | outbound_mailbox_2.READ_DATA | R | - |
| 0x2010 | outbound_mailbox_2.STATUS | R | - |
| 0x2018 | outbound_mailbox_2.ERROR_FLAGS | R | - |
| 0x2020 | outbound_mailbox_2.WIRQT | RW | - |
| 0x2028 | outbound_mailbox_2.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2030 | outbound_mailbox_2.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x2038 | outbound_mailbox_2.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x2040 | outbound_mailbox_2.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x2048 | outbound_mailbox_2.CTRL | W | - |
| 0x2800 | inbound_mailbox_2.WRITE_DATA | W | - |
| 0x2808 | inbound_mailbox_2.READ_DATA | R | - |
| 0x2810 | inbound_mailbox_2.STATUS | R | - |
| 0x2818 | inbound_mailbox_2.ERROR_FLAGS | R | - |
| 0x2820 | inbound_mailbox_2.WIRQT | RW | - |
| 0x2828 | inbound_mailbox_2.RIRQT | RW | - |
| 0x2830 | inbound_mailbox_2.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x2838 | inbound_mailbox_2.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x2840 | inbound_mailbox_2.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x2848 | inbound_mailbox_2.CTRL | W | - |
| 0x3000 | outbound_mailbox_3.WRITE_DATA | W | - |
| 0x3008 | outbound_mailbox_3.READ_DATA | R | - |
| 0x3010 | outbound_mailbox_3.STATUS | R | - |
| 0x3018 | outbound_mailbox_3.ERROR_FLAGS | R | - |
| 0x3020 | outbound_mailbox_3.WIRQT | RW | - |
| 0x3028 | outbound_mailbox_3.RIRQT | RW | - |
| 0x3030 | outbound_mailbox_3.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x3038 | outbound_mailbox_3.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x3040 | outbound_mailbox_3.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x3048 | outbound_mailbox_3.CTRL | W | - |
| 0x3800 | inbound_mailbox_3.WRITE_DATA | W | - |
| 0x3808 | inbound_mailbox_3.READ_DATA | R | - |
| 0x3810 | inbound_mailbox_3.STATUS | R | - |
| 0x3818 | inbound_mailbox_3.ERROR_FLAGS | R | - |
| 0x3820 | inbound_mailbox_3.WIRQT | RW | - |
| 0x3828 | inbound_mailbox_3.RIRQT | RW | - |
| 0x3830 | inbound_mailbox_3.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x3838 | inbound_mailbox_3.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x3840 | inbound_mailbox_3.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x3848 | inbound_mailbox_3.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x4000 | outbound_mailbox_4.WRITE_DATA | W | - |
| 0x4008 | outbound_mailbox_4.READ_DATA | R | - |
| 0x4010 | outbound_mailbox_4.STATUS | R | - |
| 0x4018 | outbound_mailbox_4.ERROR_FLAGS | R | - |
| 0x4020 | outbound_mailbox_4.WIRQT | RW | - |
| 0x4028 | outbound_mailbox_4.RIRQT | RW | - |
| 0x4030 | outbound_mailbox_4.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x4038 | outbound_mailbox_4.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x4040 | outbound_mailbox_4.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x4048 | outbound_mailbox_4.CTRL | W | - |
| 0x4800 | inbound_mailbox_4.WRITE_DATA | W | - |
| 0x4808 | inbound_mailbox_4.READ_DATA | R | - |
| 0x4810 | inbound_mailbox_4.STATUS | R | - |
| 0x4818 | inbound_mailbox_4.ERROR_FLAGS | R | - |
| 0x4820 | inbound_mailbox_4.WIRQT | RW | - |
| 0x4828 | inbound_mailbox_4.RIRQT | RW | - |
| 0x4830 | inbound_mailbox_4.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x4838 | inbound_mailbox_4.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x4840 | inbound_mailbox_4.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x4848 | inbound_mailbox_4.CTRL | W | - |
| 0x5000 | outbound_mailbox_5.WRITE_DATA | W | - |
| 0x5008 | outbound_mailbox_5.READ_DATA | R | - |
| 0x5010 | outbound_mailbox_5.STATUS | R | - |
| 0x5018 | outbound_mailbox_5.ERROR_FLAGS | R | - |
| 0x5020 | outbound_mailbox_5.WIRQT | RW | - |
| 0x5028 | outbound_mailbox_5.RIRQT | RW | - |
| 0x5030 | outbound_mailbox_5.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x5038 | outbound_mailbox_5.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x5040 | outbound_mailbox_5.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x5048 | outbound_mailbox_5.CTRL | W | - |
| 0x5800 | inbound_mailbox_5.WRITE_DATA | W | - |
| 0x5808 | inbound_mailbox_5.READ_DATA | R | - |
| 0x5810 | inbound_mailbox_5.STATUS | R | - |
| 0x5818 | inbound_mailbox_5.ERROR_FLAGS | R | - |
| 0x5820 | inbound_mailbox_5.WIRQT | RW | - |
| 0x5828 | inbound_mailbox_5.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x5830 | inbound_mailbox_5.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x5838 | inbound_mailbox_5.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x5840 | inbound_mailbox_5.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x5848 | inbound_mailbox_5.CTRL | W | - |
| 0x6000 | outbound_mailbox_6.WRITE_DATA | W | - |
| 0x6008 | outbound_mailbox_6.READ_DATA | R | - |
| 0x6010 | outbound_mailbox_6.STATUS | R | - |
| 0x6018 | outbound_mailbox_6.ERROR_FLAGS | R | - |
| 0x6020 | outbound_mailbox_6.WIRQT | RW | - |
| 0x6028 | outbound_mailbox_6.RIRQT | RW | - |
| 0x6030 | outbound_mailbox_6.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x6038 | outbound_mailbox_6.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x6040 | outbound_mailbox_6.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x6048 | outbound_mailbox_6.CTRL | W | - |
| 0x6800 | inbound_mailbox_6.WRITE_DATA | W | - |
| 0x6808 | inbound_mailbox_6.READ_DATA | R | - |
| 0x6810 | inbound_mailbox_6.STATUS | R | - |
| 0x6818 | inbound_mailbox_6.ERROR_FLAGS | R | - |
| 0x6820 | inbound_mailbox_6.WIRQT | RW | - |
| 0x6828 | inbound_mailbox_6.RIRQT | RW | - |
| 0x6830 | inbound_mailbox_6.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x6838 | inbound_mailbox_6.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x6840 | inbound_mailbox_6.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x6848 | inbound_mailbox_6.CTRL | W | - |
| 0x7000 | outbound_mailbox_7.WRITE_DATA | W | - |
| 0x7008 | outbound_mailbox_7.READ_DATA | R | - |
| 0x7010 | outbound_mailbox_7.STATUS | R | - |
| 0x7018 | outbound_mailbox_7.ERROR_FLAGS | R | - |
| 0x7020 | outbound_mailbox_7.WIRQT | RW | - |
| 0x7028 | outbound_mailbox_7.RIRQT | RW | - |
| 0x7030 | outbound_mailbox_7.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x7038 | outbound_mailbox_7.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x7040 | outbound_mailbox_7.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x7048 | outbound_mailbox_7.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x7800 | inbound_mailbox_7.WRITE_DATA | W | - |
| 0x7808 | inbound_mailbox_7.READ_DATA | R | - |
| 0x7810 | inbound_mailbox_7.STATUS | R | - |
| 0x7818 | inbound_mailbox_7.ERROR_FLAGS | R | - |
| 0x7820 | inbound_mailbox_7.WIRQT | RW | - |
| 0x7828 | inbound_mailbox_7.RIRQT | RW | - |
| 0x7830 | inbound_mailbox_7.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x7838 | inbound_mailbox_7.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x7840 | inbound_mailbox_7.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x7848 | inbound_mailbox_7.CTRL | W | - |
| 0x8000 | outbound_mailbox_8.WRITE_DATA | W | - |
| 0x8008 | outbound_mailbox_8.READ_DATA | R | - |
| 0x8010 | outbound_mailbox_8.STATUS | R | - |
| 0x8018 | outbound_mailbox_8.ERROR_FLAGS | R | - |
| 0x8020 | outbound_mailbox_8.WIRQT | RW | - |
| 0x8028 | outbound_mailbox_8.RIRQT | RW | - |
| 0x8030 | outbound_mailbox_8.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x8038 | outbound_mailbox_8.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x8040 | outbound_mailbox_8.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x8048 | outbound_mailbox_8.CTRL | W | - |
| 0x8800 | inbound_mailbox_8.WRITE_DATA | W | - |
| 0x8808 | inbound_mailbox_8.READ_DATA | R | - |
| 0x8810 | inbound_mailbox_8.STATUS | R | - |
| 0x8818 | inbound_mailbox_8.ERROR_FLAGS | R | - |
| 0x8820 | inbound_mailbox_8.WIRQT | RW | - |
| 0x8828 | inbound_mailbox_8.RIRQT | RW | - |
| 0x8830 | inbound_mailbox_8.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x8838 | inbound_mailbox_8.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x8840 | inbound_mailbox_8.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x8848 | inbound_mailbox_8.CTRL | W | - |
| 0x9000 | outbound_mailbox_9.WRITE_DATA | W | - |
| 0x9008 | outbound_mailbox_9.READ_DATA | R | - |
| 0x9010 | outbound_mailbox_9.STATUS | R | - |
| 0x9018 | outbound_mailbox_9.ERROR_FLAGS | R | - |
| 0x9020 | outbound_mailbox_9.WIRQT | RW | - |
| 0x9028 | outbound_mailbox_9.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x9030 | outbound_mailbox_9.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x9038 | outbound_mailbox_9.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x9040 | outbound_mailbox_9.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x9048 | outbound_mailbox_9.CTRL | W | - |
| 0x9800 | inbound_mailbox_9.WRITE_DATA | W | - |
| 0x9808 | inbound_mailbox_9.READ_DATA | R | - |
| 0x9810 | inbound_mailbox_9.STATUS | R | - |
| 0x9818 | inbound_mailbox_9.ERROR_FLAGS | R | - |
| 0x9820 | inbound_mailbox_9.WIRQT | RW | - |
| 0x9828 | inbound_mailbox_9.RIRQT | RW | - |
| 0x9830 | inbound_mailbox_9.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x9838 | inbound_mailbox_9.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x9840 | inbound_mailbox_9.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x9848 | inbound_mailbox_9.CTRL | W | - |
| 0xA000 | outbound_mailbox_10.WRITE_DATA | W | - |
| 0xA008 | outbound_mailbox_10.READ_DATA | R | - |
| 0xA010 | outbound_mailbox_10.STATUS | R | - |
| 0xA018 | outbound_mailbox_10.ERROR_FLAGS | R | - |
| 0xA020 | outbound_mailbox_10.WIRQT | RW | - |
| 0xA028 | outbound_mailbox_10.RIRQT | RW | - |
| 0xA030 | outbound_mailbox_10.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xA038 | outbound_mailbox_10.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xA040 | outbound_mailbox_10.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xA048 | outbound_mailbox_10.CTRL | W | - |
| 0xA800 | inbound_mailbox_10.WRITE_DATA | W | - |
| 0xA808 | inbound_mailbox_10.READ_DATA | R | - |
| 0xA810 | inbound_mailbox_10.STATUS | R | - |
| 0xA818 | inbound_mailbox_10.ERROR_FLAGS | R | - |
| 0xA820 | inbound_mailbox_10.WIRQT | RW | - |
| 0xA828 | inbound_mailbox_10.RIRQT | RW | - |
| 0xA830 | inbound_mailbox_10.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xA838 | inbound_mailbox_10.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xA840 | inbound_mailbox_10.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xA848 | inbound_mailbox_10.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0xB000 | outbound_mailbox_11.WRITE_DATA | W | - |
| 0xB008 | outbound_mailbox_11.READ_DATA | R | - |
| 0xB010 | outbound_mailbox_11.STATUS | R | - |
| 0xB018 | outbound_mailbox_11.ERROR_FLAGS | R | - |
| 0xB020 | outbound_mailbox_11.WIRQT | RW | - |
| 0xB028 | outbound_mailbox_11.RIRQT | RW | - |
| 0xB030 | outbound_mailbox_11.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xB038 | outbound_mailbox_11.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xB040 | outbound_mailbox_11.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xB048 | outbound_mailbox_11.CTRL | W | - |
| 0xB800 | inbound_mailbox_11.WRITE_DATA | W | - |
| 0xB808 | inbound_mailbox_11.READ_DATA | R | - |
| 0xB810 | inbound_mailbox_11.STATUS | R | - |
| 0xB818 | inbound_mailbox_11.ERROR_FLAGS | R | - |
| 0xB820 | inbound_mailbox_11.WIRQT | RW | - |
| 0xB828 | inbound_mailbox_11.RIRQT | RW | - |
| 0xB830 | inbound_mailbox_11.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xB838 | inbound_mailbox_11.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xB840 | inbound_mailbox_11.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xB848 | inbound_mailbox_11.CTRL | W | - |
| 0xC000 | outbound_mailbox_12.WRITE_DATA | W | - |
| 0xC008 | outbound_mailbox_12.READ_DATA | R | - |
| 0xC010 | outbound_mailbox_12.STATUS | R | - |
| 0xC018 | outbound_mailbox_12.ERROR_FLAGS | R | - |
| 0xC020 | outbound_mailbox_12.WIRQT | RW | - |
| 0xC028 | outbound_mailbox_12.RIRQT | RW | - |
| 0xC030 | outbound_mailbox_12.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xC038 | outbound_mailbox_12.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xC040 | outbound_mailbox_12.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xC048 | outbound_mailbox_12.CTRL | W | - |
| 0xC800 | inbound_mailbox_12.WRITE_DATA | W | - |
| 0xC808 | inbound_mailbox_12.READ_DATA | R | - |
| 0xC810 | inbound_mailbox_12.STATUS | R | - |
| 0xC818 | inbound_mailbox_12.ERROR_FLAGS | R | - |
| 0xC820 | inbound_mailbox_12.WIRQT | RW | - |
| 0xC828 | inbound_mailbox_12.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0xC830 | inbound_mailbox_12.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xC838 | inbound_mailbox_12.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xC840 | inbound_mailbox_12.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xC848 | inbound_mailbox_12.CTRL | W | - |
| 0xD000 | outbound_mailbox_13.WRITE_DATA | W | - |
| 0xD008 | outbound_mailbox_13.READ_DATA | R | - |
| 0xD010 | outbound_mailbox_13.STATUS | R | - |
| 0xD018 | outbound_mailbox_13.ERROR_FLAGS | R | - |
| 0xD020 | outbound_mailbox_13.WIRQT | RW | - |
| 0xD028 | outbound_mailbox_13.RIRQT | RW | - |
| 0xD030 | outbound_mailbox_13.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xD038 | outbound_mailbox_13.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xD040 | outbound_mailbox_13.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xD048 | outbound_mailbox_13.CTRL | W | - |
| 0xD800 | inbound_mailbox_13.WRITE_DATA | W | - |
| 0xD808 | inbound_mailbox_13.READ_DATA | R | - |
| 0xD810 | inbound_mailbox_13.STATUS | R | - |
| 0xD818 | inbound_mailbox_13.ERROR_FLAGS | R | - |
| 0xD820 | inbound_mailbox_13.WIRQT | RW | - |
| 0xD828 | inbound_mailbox_13.RIRQT | RW | - |
| 0xD830 | inbound_mailbox_13.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xD838 | inbound_mailbox_13.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xD840 | inbound_mailbox_13.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xD848 | inbound_mailbox_13.CTRL | W | - |
| 0xE000 | outbound_mailbox_14.WRITE_DATA | W | - |
| 0xE008 | outbound_mailbox_14.READ_DATA | R | - |
| 0xE010 | outbound_mailbox_14.STATUS | R | - |
| 0xE018 | outbound_mailbox_14.ERROR_FLAGS | R | - |
| 0xE020 | outbound_mailbox_14.WIRQT | RW | - |
| 0xE028 | outbound_mailbox_14.RIRQT | RW | - |
| 0xE030 | outbound_mailbox_14.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xE038 | outbound_mailbox_14.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xE040 | outbound_mailbox_14.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xE048 | outbound_mailbox_14.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0xE800 | inbound_mailbox_14.WRITE_DATA | W | - |
| 0xE808 | inbound_mailbox_14.READ_DATA | R | - |
| 0xE810 | inbound_mailbox_14.STATUS | R | - |
| 0xE818 | inbound_mailbox_14.ERROR_FLAGS | R | - |
| 0xE820 | inbound_mailbox_14.WIRQT | RW | - |
| 0xE828 | inbound_mailbox_14.RIRQT | RW | - |
| 0xE830 | inbound_mailbox_14.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xE838 | inbound_mailbox_14.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xE840 | inbound_mailbox_14.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xE848 | inbound_mailbox_14.CTRL | W | - |
| 0xF000 | outbound_mailbox_15.WRITE_DATA | W | - |
| 0xF008 | outbound_mailbox_15.READ_DATA | R | - |
| 0xF010 | outbound_mailbox_15.STATUS | R | - |
| 0xF018 | outbound_mailbox_15.ERROR_FLAGS | R | - |
| 0xF020 | outbound_mailbox_15.WIRQT | RW | - |
| 0xF028 | outbound_mailbox_15.RIRQT | RW | - |
| 0xF030 | outbound_mailbox_15.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xF038 | outbound_mailbox_15.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xF040 | outbound_mailbox_15.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xF048 | outbound_mailbox_15.CTRL | W | - |
| 0xF800 | inbound_mailbox_15.WRITE_DATA | W | - |
| 0xF808 | inbound_mailbox_15.READ_DATA | R | - |
| 0xF810 | inbound_mailbox_15.STATUS | R | - |
| 0xF818 | inbound_mailbox_15.ERROR_FLAGS | R | - |
| 0xF820 | inbound_mailbox_15.WIRQT | RW | - |
| 0xF828 | inbound_mailbox_15.RIRQT | RW | - |
| 0xF830 | inbound_mailbox_15.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0xF838 | inbound_mailbox_15.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0xF840 | inbound_mailbox_15.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0xF848 | inbound_mailbox_15.CTRL | W | - |
| 0x10000 | outbound_mailbox_16.WRITE_DATA | W | - |
| 0x10008 | outbound_mailbox_16.READ_DATA | R | - |
| 0x10010 | outbound_mailbox_16.STATUS | R | - |
| 0x10018 | outbound_mailbox_16.ERROR_FLAGS | R | - |
| 0x10020 | outbound_mailbox_16.WIRQT | RW | - |
| 0x10028 | outbound_mailbox_16.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x10030 | outbound_mailbox_16.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x10038 | outbound_mailbox_16.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x10040 | outbound_mailbox_16.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x10048 | outbound_mailbox_16.CTRL | W | - |
| 0x10800 | inbound_mailbox_16.WRITE_DATA | W | - |
| 0x10808 | inbound_mailbox_16.READ_DATA | R | - |
| 0x10810 | inbound_mailbox_16.STATUS | R | - |
| 0x10818 | inbound_mailbox_16.ERROR_FLAGS | R | - |
| 0x10820 | inbound_mailbox_16.WIRQT | RW | - |
| 0x10828 | inbound_mailbox_16.RIRQT | RW | - |
| 0x10830 | inbound_mailbox_16.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x10838 | inbound_mailbox_16.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x10840 | inbound_mailbox_16.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x10848 | inbound_mailbox_16.CTRL | W | - |
| 0x11000 | outbound_mailbox_17.WRITE_DATA | W | - |
| 0x11008 | outbound_mailbox_17.READ_DATA | R | - |
| 0x11010 | outbound_mailbox_17.STATUS | R | - |
| 0x11018 | outbound_mailbox_17.ERROR_FLAGS | R | - |
| 0x11020 | outbound_mailbox_17.WIRQT | RW | - |
| 0x11028 | outbound_mailbox_17.RIRQT | RW | - |
| 0x11030 | outbound_mailbox_17.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x11038 | outbound_mailbox_17.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x11040 | outbound_mailbox_17.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x11048 | outbound_mailbox_17.CTRL | W | - |
| 0x11800 | inbound_mailbox_17.WRITE_DATA | W | - |
| 0x11808 | inbound_mailbox_17.READ_DATA | R | - |
| 0x11810 | inbound_mailbox_17.STATUS | R | - |
| 0x11818 | inbound_mailbox_17.ERROR_FLAGS | R | - |
| 0x11820 | inbound_mailbox_17.WIRQT | RW | - |
| 0x11828 | inbound_mailbox_17.RIRQT | RW | - |
| 0x11830 | inbound_mailbox_17.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x11838 | inbound_mailbox_17.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x11840 | inbound_mailbox_17.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x11848 | inbound_mailbox_17.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x12000 | outbound_mailbox_18.WRITE_DATA | W | - |
| 0x12008 | outbound_mailbox_18.READ_DATA | R | - |
| 0x12010 | outbound_mailbox_18.STATUS | R | - |
| 0x12018 | outbound_mailbox_18.ERROR_FLAGS | R | - |
| 0x12020 | outbound_mailbox_18.WIRQT | RW | - |
| 0x12028 | outbound_mailbox_18.RIRQT | RW | - |
| 0x12030 | outbound_mailbox_18.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x12038 | outbound_mailbox_18.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x12040 | outbound_mailbox_18.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x12048 | outbound_mailbox_18.CTRL | W | - |
| 0x12800 | inbound_mailbox_18.WRITE_DATA | W | - |
| 0x12808 | inbound_mailbox_18.READ_DATA | R | - |
| 0x12810 | inbound_mailbox_18.STATUS | R | - |
| 0x12818 | inbound_mailbox_18.ERROR_FLAGS | R | - |
| 0x12820 | inbound_mailbox_18.WIRQT | RW | - |
| 0x12828 | inbound_mailbox_18.RIRQT | RW | - |
| 0x12830 | inbound_mailbox_18.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x12838 | inbound_mailbox_18.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x12840 | inbound_mailbox_18.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x12848 | inbound_mailbox_18.CTRL | W | - |
| 0x13000 | outbound_mailbox_19.WRITE_DATA | W | - |
| 0x13008 | outbound_mailbox_19.READ_DATA | R | - |
| 0x13010 | outbound_mailbox_19.STATUS | R | - |
| 0x13018 | outbound_mailbox_19.ERROR_FLAGS | R | - |
| 0x13020 | outbound_mailbox_19.WIRQT | RW | - |
| 0x13028 | outbound_mailbox_19.RIRQT | RW | - |
| 0x13030 | outbound_mailbox_19.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x13038 | outbound_mailbox_19.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x13040 | outbound_mailbox_19.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x13048 | outbound_mailbox_19.CTRL | W | - |
| 0x13800 | inbound_mailbox_19.WRITE_DATA | W | - |
| 0x13808 | inbound_mailbox_19.READ_DATA | R | - |
| 0x13810 | inbound_mailbox_19.STATUS | R | - |
| 0x13818 | inbound_mailbox_19.ERROR_FLAGS | R | - |
| 0x13820 | inbound_mailbox_19.WIRQT | RW | - |
| 0x13828 | inbound_mailbox_19.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x13830 | inbound_mailbox_19.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x13838 | inbound_mailbox_19.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x13840 | inbound_mailbox_19.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x13848 | inbound_mailbox_19.CTRL | W | - |
| 0x14000 | outbound_mailbox_20.WRITE_DATA | W | - |
| 0x14008 | outbound_mailbox_20.READ_DATA | R | - |
| 0x14010 | outbound_mailbox_20.STATUS | R | - |
| 0x14018 | outbound_mailbox_20.ERROR_FLAGS | R | - |
| 0x14020 | outbound_mailbox_20.WIRQT | RW | - |
| 0x14028 | outbound_mailbox_20.RIRQT | RW | - |
| 0x14030 | outbound_mailbox_20.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x14038 | outbound_mailbox_20.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x14040 | outbound_mailbox_20.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x14048 | outbound_mailbox_20.CTRL | W | - |
| 0x14800 | inbound_mailbox_20.WRITE_DATA | W | - |
| 0x14808 | inbound_mailbox_20.READ_DATA | R | - |
| 0x14810 | inbound_mailbox_20.STATUS | R | - |
| 0x14818 | inbound_mailbox_20.ERROR_FLAGS | R | - |
| 0x14820 | inbound_mailbox_20.WIRQT | RW | - |
| 0x14828 | inbound_mailbox_20.RIRQT | RW | - |
| 0x14830 | inbound_mailbox_20.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x14838 | inbound_mailbox_20.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x14840 | inbound_mailbox_20.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x14848 | inbound_mailbox_20.CTRL | W | - |
| 0x15000 | outbound_mailbox_21.WRITE_DATA | W | - |
| 0x15008 | outbound_mailbox_21.READ_DATA | R | - |
| 0x15010 | outbound_mailbox_21.STATUS | R | - |
| 0x15018 | outbound_mailbox_21.ERROR_FLAGS | R | - |
| 0x15020 | outbound_mailbox_21.WIRQT | RW | - |
| 0x15028 | outbound_mailbox_21.RIRQT | RW | - |
| 0x15030 | outbound_mailbox_21.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x15038 | outbound_mailbox_21.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x15040 | outbound_mailbox_21.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x15048 | outbound_mailbox_21.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x15800 | inbound_mailbox_21.WRITE_DATA | W | - |
| 0x15808 | inbound_mailbox_21.READ_DATA | R | - |
| 0x15810 | inbound_mailbox_21.STATUS | R | - |
| 0x15818 | inbound_mailbox_21.ERROR_FLAGS | R | - |
| 0x15820 | inbound_mailbox_21.WIRQT | RW | - |
| 0x15828 | inbound_mailbox_21.RIRQT | RW | - |
| 0x15830 | inbound_mailbox_21.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x15838 | inbound_mailbox_21.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x15840 | inbound_mailbox_21.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x15848 | inbound_mailbox_21.CTRL | W | - |
| 0x16000 | outbound_mailbox_22.WRITE_DATA | W | - |
| 0x16008 | outbound_mailbox_22.READ_DATA | R | - |
| 0x16010 | outbound_mailbox_22.STATUS | R | - |
| 0x16018 | outbound_mailbox_22.ERROR_FLAGS | R | - |
| 0x16020 | outbound_mailbox_22.WIRQT | RW | - |
| 0x16028 | outbound_mailbox_22.RIRQT | RW | - |
| 0x16030 | outbound_mailbox_22.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x16038 | outbound_mailbox_22.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x16040 | outbound_mailbox_22.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x16048 | outbound_mailbox_22.CTRL | W | - |
| 0x16800 | inbound_mailbox_22.WRITE_DATA | W | - |
| 0x16808 | inbound_mailbox_22.READ_DATA | R | - |
| 0x16810 | inbound_mailbox_22.STATUS | R | - |
| 0x16818 | inbound_mailbox_22.ERROR_FLAGS | R | - |
| 0x16820 | inbound_mailbox_22.WIRQT | RW | - |
| 0x16828 | inbound_mailbox_22.RIRQT | RW | - |
| 0x16830 | inbound_mailbox_22.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x16838 | inbound_mailbox_22.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x16840 | inbound_mailbox_22.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x16848 | inbound_mailbox_22.CTRL | W | - |
| 0x17000 | outbound_mailbox_23.WRITE_DATA | W | - |
| 0x17008 | outbound_mailbox_23.READ_DATA | R | - |
| 0x17010 | outbound_mailbox_23.STATUS | R | - |
| 0x17018 | outbound_mailbox_23.ERROR_FLAGS | R | - |
| 0x17020 | outbound_mailbox_23.WIRQT | RW | - |
| 0x17028 | outbound_mailbox_23.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x17030 | outbound_mailbox_23.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x17038 | outbound_mailbox_23.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x17040 | outbound_mailbox_23.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x17048 | outbound_mailbox_23.CTRL | W | - |
| 0x17800 | inbound_mailbox_23.WRITE_DATA | W | - |
| 0x17808 | inbound_mailbox_23.READ_DATA | R | - |
| 0x17810 | inbound_mailbox_23.STATUS | R | - |
| 0x17818 | inbound_mailbox_23.ERROR_FLAGS | R | - |
| 0x17820 | inbound_mailbox_23.WIRQT | RW | - |
| 0x17828 | inbound_mailbox_23.RIRQT | RW | - |
| 0x17830 | inbound_mailbox_23.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x17838 | inbound_mailbox_23.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x17840 | inbound_mailbox_23.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x17848 | inbound_mailbox_23.CTRL | W | - |
| 0x18000 | outbound_mailbox_24.WRITE_DATA | W | - |
| 0x18008 | outbound_mailbox_24.READ_DATA | R | - |
| 0x18010 | outbound_mailbox_24.STATUS | R | - |
| 0x18018 | outbound_mailbox_24.ERROR_FLAGS | R | - |
| 0x18020 | outbound_mailbox_24.WIRQT | RW | - |
| 0x18028 | outbound_mailbox_24.RIRQT | RW | - |
| 0x18030 | outbound_mailbox_24.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x18038 | outbound_mailbox_24.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x18040 | outbound_mailbox_24.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x18048 | outbound_mailbox_24.CTRL | W | - |
| 0x18800 | inbound_mailbox_24.WRITE_DATA | W | - |
| 0x18808 | inbound_mailbox_24.READ_DATA | R | - |
| 0x18810 | inbound_mailbox_24.STATUS | R | - |
| 0x18818 | inbound_mailbox_24.ERROR_FLAGS | R | - |
| 0x18820 | inbound_mailbox_24.WIRQT | RW | - |
| 0x18828 | inbound_mailbox_24.RIRQT | RW | - |
| 0x18830 | inbound_mailbox_24.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x18838 | inbound_mailbox_24.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x18840 | inbound_mailbox_24.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x18848 | inbound_mailbox_24.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x19000 | outbound_mailbox_25.WRITE_DATA | W | - |
| 0x19008 | outbound_mailbox_25.READ_DATA | R | - |
| 0x19010 | outbound_mailbox_25.STATUS | R | - |
| 0x19018 | outbound_mailbox_25.ERROR_FLAGS | R | - |
| 0x19020 | outbound_mailbox_25.WIRQT | RW | - |
| 0x19028 | outbound_mailbox_25.RIRQT | RW | - |
| 0x19030 | outbound_mailbox_25.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x19038 | outbound_mailbox_25.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x19040 | outbound_mailbox_25.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x19048 | outbound_mailbox_25.CTRL | W | - |
| 0x19800 | inbound_mailbox_25.WRITE_DATA | W | - |
| 0x19808 | inbound_mailbox_25.READ_DATA | R | - |
| 0x19810 | inbound_mailbox_25.STATUS | R | - |
| 0x19818 | inbound_mailbox_25.ERROR_FLAGS | R | - |
| 0x19820 | inbound_mailbox_25.WIRQT | RW | - |
| 0x19828 | inbound_mailbox_25.RIRQT | RW | - |
| 0x19830 | inbound_mailbox_25.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x19838 | inbound_mailbox_25.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x19840 | inbound_mailbox_25.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x19848 | inbound_mailbox_25.CTRL | W | - |
| 0x1A000 | outbound_mailbox_26.WRITE_DATA | W | - |
| 0x1A008 | outbound_mailbox_26.READ_DATA | R | - |
| 0x1A010 | outbound_mailbox_26.STATUS | R | - |
| 0x1A018 | outbound_mailbox_26.ERROR_FLAGS | R | - |
| 0x1A020 | outbound_mailbox_26.WIRQT | RW | - |
| 0x1A028 | outbound_mailbox_26.RIRQT | RW | - |
| 0x1A030 | outbound_mailbox_26.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1A038 | outbound_mailbox_26.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1A040 | outbound_mailbox_26.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1A048 | outbound_mailbox_26.CTRL | W | - |
| 0x1A800 | inbound_mailbox_26.WRITE_DATA | W | - |
| 0x1A808 | inbound_mailbox_26.READ_DATA | R | - |
| 0x1A810 | inbound_mailbox_26.STATUS | R | - |
| 0x1A818 | inbound_mailbox_26.ERROR_FLAGS | R | - |
| 0x1A820 | inbound_mailbox_26.WIRQT | RW | - |
| 0x1A828 | inbound_mailbox_26.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1A830 | inbound_mailbox_26.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1A838 | inbound_mailbox_26.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1A840 | inbound_mailbox_26.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1A848 | inbound_mailbox_26.CTRL | W | - |
| 0x1B000 | outbound_mailbox_27.WRITE_DATA | W | - |
| 0x1B008 | outbound_mailbox_27.READ_DATA | R | - |
| 0x1B010 | outbound_mailbox_27.STATUS | R | - |
| 0x1B018 | outbound_mailbox_27.ERROR_FLAGS | R | - |
| 0x1B020 | outbound_mailbox_27.WIRQT | RW | - |
| 0x1B028 | outbound_mailbox_27.RIRQT | RW | - |
| 0x1B030 | outbound_mailbox_27.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1B038 | outbound_mailbox_27.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1B040 | outbound_mailbox_27.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1B048 | outbound_mailbox_27.CTRL | W | - |
| 0x1B800 | inbound_mailbox_27.WRITE_DATA | W | - |
| 0x1B808 | inbound_mailbox_27.READ_DATA | R | - |
| 0x1B810 | inbound_mailbox_27.STATUS | R | - |
| 0x1B818 | inbound_mailbox_27.ERROR_FLAGS | R | - |
| 0x1B820 | inbound_mailbox_27.WIRQT | RW | - |
| 0x1B828 | inbound_mailbox_27.RIRQT | RW | - |
| 0x1B830 | inbound_mailbox_27.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1B838 | inbound_mailbox_27.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1B840 | inbound_mailbox_27.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1B848 | inbound_mailbox_27.CTRL | W | - |
| 0x1C000 | outbound_mailbox_28.WRITE_DATA | W | - |
| 0x1C008 | outbound_mailbox_28.READ_DATA | R | - |
| 0x1C010 | outbound_mailbox_28.STATUS | R | - |
| 0x1C018 | outbound_mailbox_28.ERROR_FLAGS | R | - |
| 0x1C020 | outbound_mailbox_28.WIRQT | RW | - |
| 0x1C028 | outbound_mailbox_28.RIRQT | RW | - |
| 0x1C030 | outbound_mailbox_28.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1C038 | outbound_mailbox_28.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1C040 | outbound_mailbox_28.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1C048 | outbound_mailbox_28.CTRL | W | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1C800 | inbound_mailbox_28.WRITE_DATA | W | - |
| 0x1C808 | inbound_mailbox_28.READ_DATA | R | - |
| 0x1C810 | inbound_mailbox_28.STATUS | R | - |
| 0x1C818 | inbound_mailbox_28.ERROR_FLAGS | R | - |
| 0x1C820 | inbound_mailbox_28.WIRQT | RW | - |
| 0x1C828 | inbound_mailbox_28.RIRQT | RW | - |
| 0x1C830 | inbound_mailbox_28.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1C838 | inbound_mailbox_28.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1C840 | inbound_mailbox_28.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1C848 | inbound_mailbox_28.CTRL | W | - |
| 0x1D000 | outbound_mailbox_29.WRITE_DATA | W | - |
| 0x1D008 | outbound_mailbox_29.READ_DATA | R | - |
| 0x1D010 | outbound_mailbox_29.STATUS | R | - |
| 0x1D018 | outbound_mailbox_29.ERROR_FLAGS | R | - |
| 0x1D020 | outbound_mailbox_29.WIRQT | RW | - |
| 0x1D028 | outbound_mailbox_29.RIRQT | RW | - |
| 0x1D030 | outbound_mailbox_29.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1D038 | outbound_mailbox_29.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1D040 | outbound_mailbox_29.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1D048 | outbound_mailbox_29.CTRL | W | - |
| 0x1D800 | inbound_mailbox_29.WRITE_DATA | W | - |
| 0x1D808 | inbound_mailbox_29.READ_DATA | R | - |
| 0x1D810 | inbound_mailbox_29.STATUS | R | - |
| 0x1D818 | inbound_mailbox_29.ERROR_FLAGS | R | - |
| 0x1D820 | inbound_mailbox_29.WIRQT | RW | - |
| 0x1D828 | inbound_mailbox_29.RIRQT | RW | - |
| 0x1D830 | inbound_mailbox_29.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1D838 | inbound_mailbox_29.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1D840 | inbound_mailbox_29.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1D848 | inbound_mailbox_29.CTRL | W | - |
| 0x1E000 | outbound_mailbox_30.WRITE_DATA | W | - |
| 0x1E008 | outbound_mailbox_30.READ_DATA | R | - |
| 0x1E010 | outbound_mailbox_30.STATUS | R | - |
| 0x1E018 | outbound_mailbox_30.ERROR_FLAGS | R | - |
| 0x1E020 | outbound_mailbox_30.WIRQT | RW | - |
| 0x1E028 | outbound_mailbox_30.RIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1E030 | outbound_mailbox_30.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1E038 | outbound_mailbox_30.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1E040 | outbound_mailbox_30.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1E048 | outbound_mailbox_30.CTRL | W | - |
| 0x1E800 | inbound_mailbox_30.WRITE_DATA | W | - |
| 0x1E808 | inbound_mailbox_30.READ_DATA | R | - |
| 0x1E810 | inbound_mailbox_30.STATUS | R | - |
| 0x1E818 | inbound_mailbox_30.ERROR_FLAGS | R | - |
| 0x1E820 | inbound_mailbox_30.WIRQT | RW | - |
| 0x1E828 | inbound_mailbox_30.RIRQT | RW | - |
| 0x1E830 | inbound_mailbox_30.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1E838 | inbound_mailbox_30.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1E840 | inbound_mailbox_30.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1E848 | inbound_mailbox_30.CTRL | W | - |
| 0x1F000 | outbound_mailbox_31.WRITE_DATA | W | - |
| 0x1F008 | outbound_mailbox_31.READ_DATA | R | - |
| 0x1F010 | outbound_mailbox_31.STATUS | R | - |
| 0x1F018 | outbound_mailbox_31.ERROR_FLAGS | R | - |
| 0x1F020 | outbound_mailbox_31.WIRQT | RW | - |
| 0x1F028 | outbound_mailbox_31.RIRQT | RW | - |
| 0x1F030 | outbound_mailbox_31.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1F038 | outbound_mailbox_31.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1F040 | outbound_mailbox_31.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1F048 | outbound_mailbox_31.CTRL | W | - |
| 0x1F800 | inbound_mailbox_31.WRITE_DATA | W | - |
| 0x1F808 | inbound_mailbox_31.READ_DATA | R | - |
| 0x1F810 | inbound_mailbox_31.STATUS | R | - |
| 0x1F818 | inbound_mailbox_31.ERROR_FLAGS | R | - |
| 0x1F820 | inbound_mailbox_31.WIRQT | RW | - |
| 0x1F828 | inbound_mailbox_31.RIRQT | RW | - |
| 0x1F830 | inbound_mailbox_31.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1F838 | inbound_mailbox_31.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1F840 | inbound_mailbox_31.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1F848 | inbound_mailbox_31.CTRL | W | - |

### outbound_mailbox_0.WRITE_DATA

*Table 1072. axil_mailbox_smc_wrap.outbound_mailbox_0.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_0.READ_DATA

*Table 1073. axil_mailbox_smc_wrap.outbound_mailbox_0.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_0.STATUS

*Table 1074. axil_mailbox_smc_wrap.outbound_mailbox_0.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_0.ERROR_FLAGS

*Table 1075. axil_mailbox_smc_wrap.outbound_mailbox_0.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_0.WIRQT

*Table 1076. axil_mailbox_smc_wrap.outbound_mailbox_0.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_0.RIRQT

*Table 1077. axil_mailbox_smc_wrap.outbound_mailbox_0.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_0.IRQS

*Table 1078. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_0.IRQEN

*Table 1079. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_0.IRQP

*Table 1080. axil_mailbox_smc_wrap.outbound_mailbox_0.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_0.CTRL

*Table 1081. axil_mailbox_smc_wrap.outbound_mailbox_0.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_0.WRITE_DATA

*Table 1082. axil_mailbox_smc_wrap.inbound_mailbox_0.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_0.READ_DATA

*Table 1083. axil_mailbox_smc_wrap.inbound_mailbox_0.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_0.STATUS

*Table 1084. axil_mailbox_smc_wrap.inbound_mailbox_0.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_0.ERROR_FLAGS

*Table 1085. axil_mailbox_smc_wrap.inbound_mailbox_0.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_0.WIRQT

*Table 1086. axil_mailbox_smc_wrap.inbound_mailbox_0.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_0.RIRQT

*Table 1087. axil_mailbox_smc_wrap.inbound_mailbox_0.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_0.IRQS

*Table 1088. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_0.IRQEN

*Table 1089. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_0.IRQP

*Table 1090. axil_mailbox_smc_wrap.inbound_mailbox_0.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_0.CTRL

*Table 1091. axil_mailbox_smc_wrap.inbound_mailbox_0.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_1.WRITE_DATA

*Table 1092. axil_mailbox_smc_wrap.outbound_mailbox_1.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_1.READ_DATA

*Table 1093. axil_mailbox_smc_wrap.outbound_mailbox_1.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_1.STATUS

*Table 1094. axil_mailbox_smc_wrap.outbound_mailbox_1.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_1.ERROR_FLAGS

*Table 1095. axil_mailbox_smc_wrap.outbound_mailbox_1.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_1.WIRQT

*Table 1096. axil_mailbox_smc_wrap.outbound_mailbox_1.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_1.RIRQT

*Table 1097. axil_mailbox_smc_wrap.outbound_mailbox_1.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_1.IRQS

*Table 1098. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_1.IRQEN

*Table 1099. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_1.IRQP

*Table 1100. axil_mailbox_smc_wrap.outbound_mailbox_1.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_1.CTRL

*Table 1101. axil_mailbox_smc_wrap.outbound_mailbox_1.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_1.WRITE_DATA

*Table 1102. axil_mailbox_smc_wrap.inbound_mailbox_1.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_1.READ_DATA

*Table 1103. axil_mailbox_smc_wrap.inbound_mailbox_1.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_1.STATUS

*Table 1104. axil_mailbox_smc_wrap.inbound_mailbox_1.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_1.ERROR_FLAGS

*Table 1105. axil_mailbox_smc_wrap.inbound_mailbox_1.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_1.WIRQT

*Table 1106. axil_mailbox_smc_wrap.inbound_mailbox_1.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_1.RIRQT

*Table 1107. axil_mailbox_smc_wrap.inbound_mailbox_1.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_1.IRQS

*Table 1108. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_1.IRQEN

*Table 1109. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_1.IRQP

*Table 1110. axil_mailbox_smc_wrap.inbound_mailbox_1.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_1.CTRL

*Table 1111. axil_mailbox_smc_wrap.inbound_mailbox_1.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_2.WRITE_DATA

*Table 1112. axil_mailbox_smc_wrap.outbound_mailbox_2.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_2.READ_DATA

*Table 1113. axil_mailbox_smc_wrap.outbound_mailbox_2.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_2.STATUS

*Table 1114. axil_mailbox_smc_wrap.outbound_mailbox_2.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_2.ERROR_FLAGS

*Table 1115. axil_mailbox_smc_wrap.outbound_mailbox_2.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_2.WIRQT

*Table 1116. axil_mailbox_smc_wrap.outbound_mailbox_2.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_2.RIRQT

*Table 1117. axil_mailbox_smc_wrap.outbound_mailbox_2.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_2.IRQS

*Table 1118. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_2.IRQEN

*Table 1119. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_2.IRQP

*Table 1120. axil_mailbox_smc_wrap.outbound_mailbox_2.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_2.CTRL

*Table 1121. axil_mailbox_smc_wrap.outbound_mailbox_2.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_2.WRITE_DATA

*Table 1122. axil_mailbox_smc_wrap.inbound_mailbox_2.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_2.READ_DATA

*Table 1123. axil_mailbox_smc_wrap.inbound_mailbox_2.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_2.STATUS

*Table 1124. axil_mailbox_smc_wrap.inbound_mailbox_2.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_2.ERROR_FLAGS

*Table 1125. axil_mailbox_smc_wrap.inbound_mailbox_2.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_2.WIRQT

*Table 1126. axil_mailbox_smc_wrap.inbound_mailbox_2.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_2.RIRQT

*Table 1127. axil_mailbox_smc_wrap.inbound_mailbox_2.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_2.IRQS

*Table 1128. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_2.IRQEN

*Table 1129. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_2.IRQP

*Table 1130. axil_mailbox_smc_wrap.inbound_mailbox_2.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_2.CTRL

*Table 1131. axil_mailbox_smc_wrap.inbound_mailbox_2.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_3.WRITE_DATA

*Table 1132. axil_mailbox_smc_wrap.outbound_mailbox_3.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_3.READ_DATA

*Table 1133. axil_mailbox_smc_wrap.outbound_mailbox_3.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_3.STATUS

*Table 1134. axil_mailbox_smc_wrap.outbound_mailbox_3.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_3.ERROR_FLAGS

*Table 1135. axil_mailbox_smc_wrap.outbound_mailbox_3.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_3.WIRQT

*Table 1136. axil_mailbox_smc_wrap.outbound_mailbox_3.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_3.RIRQT

*Table 1137. axil_mailbox_smc_wrap.outbound_mailbox_3.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_3.IRQS

*Table 1138. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_3.IRQEN

*Table 1139. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_3.IRQP

*Table 1140. axil_mailbox_smc_wrap.outbound_mailbox_3.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_3.CTRL

*Table 1141. axil_mailbox_smc_wrap.outbound_mailbox_3.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_3.WRITE_DATA

*Table 1142. axil_mailbox_smc_wrap.inbound_mailbox_3.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_3.READ_DATA

*Table 1143. axil_mailbox_smc_wrap.inbound_mailbox_3.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_3.STATUS

*Table 1144. axil_mailbox_smc_wrap.inbound_mailbox_3.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_3.ERROR_FLAGS

*Table 1145. axil_mailbox_smc_wrap.inbound_mailbox_3.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_3.WIRQT

*Table 1146. axil_mailbox_smc_wrap.inbound_mailbox_3.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_3.RIRQT

*Table 1147. axil_mailbox_smc_wrap.inbound_mailbox_3.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_3.IRQS

*Table 1148. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_3.IRQEN

*Table 1149. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_3.IRQP

*Table 1150. axil_mailbox_smc_wrap.inbound_mailbox_3.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_3.CTRL

*Table 1151. axil_mailbox_smc_wrap.inbound_mailbox_3.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_4.WRITE_DATA

*Table 1152. axil_mailbox_smc_wrap.outbound_mailbox_4.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_4.READ_DATA

*Table 1153. axil_mailbox_smc_wrap.outbound_mailbox_4.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_4.STATUS

*Table 1154. axil_mailbox_smc_wrap.outbound_mailbox_4.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_4.ERROR_FLAGS

*Table 1155. axil_mailbox_smc_wrap.outbound_mailbox_4.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_4.WIRQT

*Table 1156. axil_mailbox_smc_wrap.outbound_mailbox_4.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_4.RIRQT

*Table 1157. axil_mailbox_smc_wrap.outbound_mailbox_4.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_4.IRQS

*Table 1158. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_4.IRQEN

*Table 1159. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_4.IRQP

*Table 1160. axil_mailbox_smc_wrap.outbound_mailbox_4.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_4.CTRL

*Table 1161. axil_mailbox_smc_wrap.outbound_mailbox_4.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_4.WRITE_DATA

*Table 1162. axil_mailbox_smc_wrap.inbound_mailbox_4.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_4.READ_DATA

*Table 1163. axil_mailbox_smc_wrap.inbound_mailbox_4.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_4.STATUS

*Table 1164. axil_mailbox_smc_wrap.inbound_mailbox_4.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_4.ERROR_FLAGS

*Table 1165. axil_mailbox_smc_wrap.inbound_mailbox_4.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_4.WIRQT

*Table 1166. axil_mailbox_smc_wrap.inbound_mailbox_4.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_4.RIRQT

*Table 1167. axil_mailbox_smc_wrap.inbound_mailbox_4.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_4.IRQS

*Table 1168. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_4.IRQEN

*Table 1169. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_4.IRQP

*Table 1170. axil_mailbox_smc_wrap.inbound_mailbox_4.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_4.CTRL

*Table 1171. axil_mailbox_smc_wrap.inbound_mailbox_4.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_5.WRITE_DATA

*Table 1172. axil_mailbox_smc_wrap.outbound_mailbox_5.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_5.READ_DATA

*Table 1173. axil_mailbox_smc_wrap.outbound_mailbox_5.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_5.STATUS

*Table 1174. axil_mailbox_smc_wrap.outbound_mailbox_5.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_5.ERROR_FLAGS

*Table 1175. axil_mailbox_smc_wrap.outbound_mailbox_5.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_5.WIRQT

*Table 1176. axil_mailbox_smc_wrap.outbound_mailbox_5.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_5.RIRQT

*Table 1177. axil_mailbox_smc_wrap.outbound_mailbox_5.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_5.IRQS

*Table 1178. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_5.IRQEN

*Table 1179. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_5.IRQP

*Table 1180. axil_mailbox_smc_wrap.outbound_mailbox_5.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_5.CTRL

*Table 1181. axil_mailbox_smc_wrap.outbound_mailbox_5.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_5.WRITE_DATA

*Table 1182. axil_mailbox_smc_wrap.inbound_mailbox_5.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_5.READ_DATA

*Table 1183. axil_mailbox_smc_wrap.inbound_mailbox_5.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_5.STATUS

*Table 1184. axil_mailbox_smc_wrap.inbound_mailbox_5.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_5.ERROR_FLAGS

*Table 1185. axil_mailbox_smc_wrap.inbound_mailbox_5.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_5.WIRQT

*Table 1186. axil_mailbox_smc_wrap.inbound_mailbox_5.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_5.RIRQT

*Table 1187. axil_mailbox_smc_wrap.inbound_mailbox_5.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_5.IRQS

*Table 1188. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_5.IRQEN

*Table 1189. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_5.IRQP

*Table 1190. axil_mailbox_smc_wrap.inbound_mailbox_5.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_5.CTRL

*Table 1191. axil_mailbox_smc_wrap.inbound_mailbox_5.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_6.WRITE_DATA

*Table 1192. axil_mailbox_smc_wrap.outbound_mailbox_6.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_6.READ_DATA

*Table 1193. axil_mailbox_smc_wrap.outbound_mailbox_6.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_6.STATUS

*Table 1194. axil_mailbox_smc_wrap.outbound_mailbox_6.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_6.ERROR_FLAGS

*Table 1195. axil_mailbox_smc_wrap.outbound_mailbox_6.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_6.WIRQT

*Table 1196. axil_mailbox_smc_wrap.outbound_mailbox_6.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_6.RIRQT

*Table 1197. axil_mailbox_smc_wrap.outbound_mailbox_6.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_6.IRQS

*Table 1198. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_6.IRQEN

*Table 1199. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_6.IRQP

*Table 1200. axil_mailbox_smc_wrap.outbound_mailbox_6.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_6.CTRL

*Table 1201. axil_mailbox_smc_wrap.outbound_mailbox_6.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_6.WRITE_DATA

*Table 1202. axil_mailbox_smc_wrap.inbound_mailbox_6.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_6.READ_DATA

*Table 1203. axil_mailbox_smc_wrap.inbound_mailbox_6.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_6.STATUS

*Table 1204. axil_mailbox_smc_wrap.inbound_mailbox_6.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_6.ERROR_FLAGS

*Table 1205. axil_mailbox_smc_wrap.inbound_mailbox_6.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_6.WIRQT

*Table 1206. axil_mailbox_smc_wrap.inbound_mailbox_6.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_6.RIRQT

*Table 1207. axil_mailbox_smc_wrap.inbound_mailbox_6.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_6.IRQS

*Table 1208. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_6.IRQEN

*Table 1209. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_6.IRQP

*Table 1210. axil_mailbox_smc_wrap.inbound_mailbox_6.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_6.CTRL

*Table 1211. axil_mailbox_smc_wrap.inbound_mailbox_6.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_7.WRITE_DATA

*Table 1212. axil_mailbox_smc_wrap.outbound_mailbox_7.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_7.READ_DATA

*Table 1213. axil_mailbox_smc_wrap.outbound_mailbox_7.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_7.STATUS

*Table 1214. axil_mailbox_smc_wrap.outbound_mailbox_7.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_7.ERROR_FLAGS

*Table 1215. axil_mailbox_smc_wrap.outbound_mailbox_7.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_7.WIRQT

*Table 1216. axil_mailbox_smc_wrap.outbound_mailbox_7.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_7.RIRQT

*Table 1217. axil_mailbox_smc_wrap.outbound_mailbox_7.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_7.IRQS

*Table 1218. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_7.IRQEN

*Table 1219. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_7.IRQP

*Table 1220. axil_mailbox_smc_wrap.outbound_mailbox_7.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_7.CTRL

*Table 1221. axil_mailbox_smc_wrap.outbound_mailbox_7.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_7.WRITE_DATA

*Table 1222. axil_mailbox_smc_wrap.inbound_mailbox_7.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_7.READ_DATA

*Table 1223. axil_mailbox_smc_wrap.inbound_mailbox_7.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_7.STATUS

*Table 1224. axil_mailbox_smc_wrap.inbound_mailbox_7.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_7.ERROR_FLAGS

*Table 1225. axil_mailbox_smc_wrap.inbound_mailbox_7.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_7.WIRQT

*Table 1226. axil_mailbox_smc_wrap.inbound_mailbox_7.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_7.RIRQT

*Table 1227. axil_mailbox_smc_wrap.inbound_mailbox_7.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_7.IRQS

*Table 1228. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_7.IRQEN

*Table 1229. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_7.IRQP

*Table 1230. axil_mailbox_smc_wrap.inbound_mailbox_7.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_7.CTRL

*Table 1231. axil_mailbox_smc_wrap.inbound_mailbox_7.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_8.WRITE_DATA

*Table 1232. axil_mailbox_smc_wrap.outbound_mailbox_8.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_8.READ_DATA

*Table 1233. axil_mailbox_smc_wrap.outbound_mailbox_8.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_8.STATUS

*Table 1234. axil_mailbox_smc_wrap.outbound_mailbox_8.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_8.ERROR_FLAGS

*Table 1235. axil_mailbox_smc_wrap.outbound_mailbox_8.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_8.WIRQT

*Table 1236. axil_mailbox_smc_wrap.outbound_mailbox_8.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_8.RIRQT

*Table 1237. axil_mailbox_smc_wrap.outbound_mailbox_8.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_8.IRQS

*Table 1238. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_8.IRQEN

*Table 1239. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_8.IRQP

*Table 1240. axil_mailbox_smc_wrap.outbound_mailbox_8.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_8.CTRL

*Table 1241. axil_mailbox_smc_wrap.outbound_mailbox_8.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_8.WRITE_DATA

*Table 1242. axil_mailbox_smc_wrap.inbound_mailbox_8.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_8.READ_DATA

*Table 1243. axil_mailbox_smc_wrap.inbound_mailbox_8.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_8.STATUS

*Table 1244. axil_mailbox_smc_wrap.inbound_mailbox_8.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_8.ERROR_FLAGS

*Table 1245. axil_mailbox_smc_wrap.inbound_mailbox_8.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_8.WIRQT

*Table 1246. axil_mailbox_smc_wrap.inbound_mailbox_8.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_8.RIRQT

*Table 1247. axil_mailbox_smc_wrap.inbound_mailbox_8.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_8.IRQS

*Table 1248. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_8.IRQEN

*Table 1249. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_8.IRQP

*Table 1250. axil_mailbox_smc_wrap.inbound_mailbox_8.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_8.CTRL

*Table 1251. axil_mailbox_smc_wrap.inbound_mailbox_8.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_9.WRITE_DATA

*Table 1252. axil_mailbox_smc_wrap.outbound_mailbox_9.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_9.READ_DATA

*Table 1253. axil_mailbox_smc_wrap.outbound_mailbox_9.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_9.STATUS

*Table 1254. axil_mailbox_smc_wrap.outbound_mailbox_9.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_9.ERROR_FLAGS

*Table 1255. axil_mailbox_smc_wrap.outbound_mailbox_9.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_9.WIRQT

*Table 1256. axil_mailbox_smc_wrap.outbound_mailbox_9.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_9.RIRQT

*Table 1257. axil_mailbox_smc_wrap.outbound_mailbox_9.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_9.IRQS

*Table 1258. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_9.IRQEN

*Table 1259. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_9.IRQP

*Table 1260. axil_mailbox_smc_wrap.outbound_mailbox_9.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_9.CTRL

*Table 1261. axil_mailbox_smc_wrap.outbound_mailbox_9.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_9.WRITE_DATA

*Table 1262. axil_mailbox_smc_wrap.inbound_mailbox_9.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_9.READ_DATA

*Table 1263. axil_mailbox_smc_wrap.inbound_mailbox_9.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_9.STATUS

*Table 1264. axil_mailbox_smc_wrap.inbound_mailbox_9.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_9.ERROR_FLAGS

*Table 1265. axil_mailbox_smc_wrap.inbound_mailbox_9.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_9.WIRQT

*Table 1266. axil_mailbox_smc_wrap.inbound_mailbox_9.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_9.RIRQT

*Table 1267. axil_mailbox_smc_wrap.inbound_mailbox_9.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_9.IRQS

*Table 1268. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_9.IRQEN

*Table 1269. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_9.IRQP

*Table 1270. axil_mailbox_smc_wrap.inbound_mailbox_9.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_9.CTRL

*Table 1271. axil_mailbox_smc_wrap.inbound_mailbox_9.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_10.WRITE_DATA

*Table 1272. axil_mailbox_smc_wrap.outbound_mailbox_10.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_10.READ_DATA

*Table 1273. axil_mailbox_smc_wrap.outbound_mailbox_10.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_10.STATUS

*Table 1274. axil_mailbox_smc_wrap.outbound_mailbox_10.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_10.ERROR_FLAGS

*Table 1275. axil_mailbox_smc_wrap.outbound_mailbox_10.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_10.WIRQT

*Table 1276. axil_mailbox_smc_wrap.outbound_mailbox_10.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_10.RIRQT

*Table 1277. axil_mailbox_smc_wrap.outbound_mailbox_10.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_10.IRQS

*Table 1278. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_10.IRQEN

*Table 1279. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_10.IRQP

*Table 1280. axil_mailbox_smc_wrap.outbound_mailbox_10.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_10.CTRL

*Table 1281. axil_mailbox_smc_wrap.outbound_mailbox_10.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_10.WRITE_DATA

*Table 1282. axil_mailbox_smc_wrap.inbound_mailbox_10.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_10.READ_DATA

*Table 1283. axil_mailbox_smc_wrap.inbound_mailbox_10.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_10.STATUS

*Table 1284. axil_mailbox_smc_wrap.inbound_mailbox_10.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_10.ERROR_FLAGS

*Table 1285. axil_mailbox_smc_wrap.inbound_mailbox_10.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_10.WIRQT

*Table 1286. axil_mailbox_smc_wrap.inbound_mailbox_10.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_10.RIRQT

*Table 1287. axil_mailbox_smc_wrap.inbound_mailbox_10.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_10.IRQS

*Table 1288. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_10.IRQEN

*Table 1289. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_10.IRQP

*Table 1290. axil_mailbox_smc_wrap.inbound_mailbox_10.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_10.CTRL

*Table 1291. axil_mailbox_smc_wrap.inbound_mailbox_10.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_11.WRITE_DATA

*Table 1292. axil_mailbox_smc_wrap.outbound_mailbox_11.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_11.READ_DATA

*Table 1293. axil_mailbox_smc_wrap.outbound_mailbox_11.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_11.STATUS

*Table 1294. axil_mailbox_smc_wrap.outbound_mailbox_11.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_11.ERROR_FLAGS

*Table 1295. axil_mailbox_smc_wrap.outbound_mailbox_11.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_11.WIRQT

*Table 1296. axil_mailbox_smc_wrap.outbound_mailbox_11.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_11.RIRQT

*Table 1297. axil_mailbox_smc_wrap.outbound_mailbox_11.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_11.IRQS

*Table 1298. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_11.IRQEN

*Table 1299. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_11.IRQP

*Table 1300. axil_mailbox_smc_wrap.outbound_mailbox_11.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_11.CTRL

*Table 1301. axil_mailbox_smc_wrap.outbound_mailbox_11.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_11.WRITE_DATA

*Table 1302. axil_mailbox_smc_wrap.inbound_mailbox_11.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_11.READ_DATA

*Table 1303. axil_mailbox_smc_wrap.inbound_mailbox_11.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_11.STATUS

*Table 1304. axil_mailbox_smc_wrap.inbound_mailbox_11.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_11.ERROR_FLAGS

*Table 1305. axil_mailbox_smc_wrap.inbound_mailbox_11.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_11.WIRQT

*Table 1306. axil_mailbox_smc_wrap.inbound_mailbox_11.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_11.RIRQT

*Table 1307. axil_mailbox_smc_wrap.inbound_mailbox_11.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_11.IRQS

*Table 1308. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_11.IRQEN

*Table 1309. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_11.IRQP

*Table 1310. axil_mailbox_smc_wrap.inbound_mailbox_11.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_11.CTRL

*Table 1311. axil_mailbox_smc_wrap.inbound_mailbox_11.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_12.WRITE_DATA

*Table 1312. axil_mailbox_smc_wrap.outbound_mailbox_12.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_12.READ_DATA

*Table 1313. axil_mailbox_smc_wrap.outbound_mailbox_12.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_12.STATUS

*Table 1314. axil_mailbox_smc_wrap.outbound_mailbox_12.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_12.ERROR_FLAGS

*Table 1315. axil_mailbox_smc_wrap.outbound_mailbox_12.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_12.WIRQT

*Table 1316. axil_mailbox_smc_wrap.outbound_mailbox_12.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_12.RIRQT

*Table 1317. axil_mailbox_smc_wrap.outbound_mailbox_12.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_12.IRQS

*Table 1318. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_12.IRQEN

*Table 1319. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_12.IRQP

*Table 1320. axil_mailbox_smc_wrap.outbound_mailbox_12.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_12.CTRL

*Table 1321. axil_mailbox_smc_wrap.outbound_mailbox_12.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_12.WRITE_DATA

*Table 1322. axil_mailbox_smc_wrap.inbound_mailbox_12.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_12.READ_DATA

*Table 1323. axil_mailbox_smc_wrap.inbound_mailbox_12.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_12.STATUS

*Table 1324. axil_mailbox_smc_wrap.inbound_mailbox_12.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_12.ERROR_FLAGS

*Table 1325. axil_mailbox_smc_wrap.inbound_mailbox_12.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_12.WIRQT

*Table 1326. axil_mailbox_smc_wrap.inbound_mailbox_12.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_12.RIRQT

*Table 1327. axil_mailbox_smc_wrap.inbound_mailbox_12.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_12.IRQS

*Table 1328. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_12.IRQEN

*Table 1329. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_12.IRQP

*Table 1330. axil_mailbox_smc_wrap.inbound_mailbox_12.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_12.CTRL

*Table 1331. axil_mailbox_smc_wrap.inbound_mailbox_12.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_13.WRITE_DATA

*Table 1332. axil_mailbox_smc_wrap.outbound_mailbox_13.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_13.READ_DATA

*Table 1333. axil_mailbox_smc_wrap.outbound_mailbox_13.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_13.STATUS

*Table 1334. axil_mailbox_smc_wrap.outbound_mailbox_13.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_13.ERROR_FLAGS

*Table 1335. axil_mailbox_smc_wrap.outbound_mailbox_13.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_13.WIRQT

*Table 1336. axil_mailbox_smc_wrap.outbound_mailbox_13.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_13.RIRQT

*Table 1337. axil_mailbox_smc_wrap.outbound_mailbox_13.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_13.IRQS

*Table 1338. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_13.IRQEN

*Table 1339. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_13.IRQP

*Table 1340. axil_mailbox_smc_wrap.outbound_mailbox_13.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_13.CTRL

*Table 1341. axil_mailbox_smc_wrap.outbound_mailbox_13.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_13.WRITE_DATA

*Table 1342. axil_mailbox_smc_wrap.inbound_mailbox_13.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_13.READ_DATA

*Table 1343. axil_mailbox_smc_wrap.inbound_mailbox_13.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_13.STATUS

*Table 1344. axil_mailbox_smc_wrap.inbound_mailbox_13.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_13.ERROR_FLAGS

*Table 1345. axil_mailbox_smc_wrap.inbound_mailbox_13.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_13.WIRQT

*Table 1346. axil_mailbox_smc_wrap.inbound_mailbox_13.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_13.RIRQT

*Table 1347. axil_mailbox_smc_wrap.inbound_mailbox_13.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_13.IRQS

*Table 1348. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_13.IRQEN

*Table 1349. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_13.IRQP

*Table 1350. axil_mailbox_smc_wrap.inbound_mailbox_13.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_13.CTRL

*Table 1351. axil_mailbox_smc_wrap.inbound_mailbox_13.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_14.WRITE_DATA

*Table 1352. axil_mailbox_smc_wrap.outbound_mailbox_14.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_14.READ_DATA

*Table 1353. axil_mailbox_smc_wrap.outbound_mailbox_14.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_14.STATUS

*Table 1354. axil_mailbox_smc_wrap.outbound_mailbox_14.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_14.ERROR_FLAGS

*Table 1355. axil_mailbox_smc_wrap.outbound_mailbox_14.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_14.WIRQT

*Table 1356. axil_mailbox_smc_wrap.outbound_mailbox_14.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_14.RIRQT

*Table 1357. axil_mailbox_smc_wrap.outbound_mailbox_14.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_14.IRQS

*Table 1358. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_14.IRQEN

*Table 1359. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_14.IRQP

*Table 1360. axil_mailbox_smc_wrap.outbound_mailbox_14.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_14.CTRL

*Table 1361. axil_mailbox_smc_wrap.outbound_mailbox_14.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_14.WRITE_DATA

*Table 1362. axil_mailbox_smc_wrap.inbound_mailbox_14.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_14.READ_DATA

*Table 1363. axil_mailbox_smc_wrap.inbound_mailbox_14.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_14.STATUS

*Table 1364. axil_mailbox_smc_wrap.inbound_mailbox_14.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_14.ERROR_FLAGS

*Table 1365. axil_mailbox_smc_wrap.inbound_mailbox_14.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_14.WIRQT

*Table 1366. axil_mailbox_smc_wrap.inbound_mailbox_14.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_14.RIRQT

*Table 1367. axil_mailbox_smc_wrap.inbound_mailbox_14.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_14.IRQS

*Table 1368. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_14.IRQEN

*Table 1369. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_14.IRQP

*Table 1370. axil_mailbox_smc_wrap.inbound_mailbox_14.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_14.CTRL

*Table 1371. axil_mailbox_smc_wrap.inbound_mailbox_14.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_15.WRITE_DATA

*Table 1372. axil_mailbox_smc_wrap.outbound_mailbox_15.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_15.READ_DATA

*Table 1373. axil_mailbox_smc_wrap.outbound_mailbox_15.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_15.STATUS

*Table 1374. axil_mailbox_smc_wrap.outbound_mailbox_15.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_15.ERROR_FLAGS

*Table 1375. axil_mailbox_smc_wrap.outbound_mailbox_15.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_15.WIRQT

*Table 1376. axil_mailbox_smc_wrap.outbound_mailbox_15.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_15.RIRQT

*Table 1377. axil_mailbox_smc_wrap.outbound_mailbox_15.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_15.IRQS

*Table 1378. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_15.IRQEN

*Table 1379. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_15.IRQP

*Table 1380. axil_mailbox_smc_wrap.outbound_mailbox_15.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_15.CTRL

*Table 1381. axil_mailbox_smc_wrap.outbound_mailbox_15.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_15.WRITE_DATA

*Table 1382. axil_mailbox_smc_wrap.inbound_mailbox_15.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_15.READ_DATA

*Table 1383. axil_mailbox_smc_wrap.inbound_mailbox_15.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_15.STATUS

*Table 1384. axil_mailbox_smc_wrap.inbound_mailbox_15.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_15.ERROR_FLAGS

*Table 1385. axil_mailbox_smc_wrap.inbound_mailbox_15.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_15.WIRQT

*Table 1386. axil_mailbox_smc_wrap.inbound_mailbox_15.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_15.RIRQT

*Table 1387. axil_mailbox_smc_wrap.inbound_mailbox_15.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_15.IRQS

*Table 1388. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_15.IRQEN

*Table 1389. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_15.IRQP

*Table 1390. axil_mailbox_smc_wrap.inbound_mailbox_15.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_15.CTRL

*Table 1391. axil_mailbox_smc_wrap.inbound_mailbox_15.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_16.WRITE_DATA

*Table 1392. axil_mailbox_smc_wrap.outbound_mailbox_16.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_16.READ_DATA

*Table 1393. axil_mailbox_smc_wrap.outbound_mailbox_16.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_16.STATUS

*Table 1394. axil_mailbox_smc_wrap.outbound_mailbox_16.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_16.ERROR_FLAGS

*Table 1395. axil_mailbox_smc_wrap.outbound_mailbox_16.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_16.WIRQT

*Table 1396. axil_mailbox_smc_wrap.outbound_mailbox_16.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_16.RIRQT

*Table 1397. axil_mailbox_smc_wrap.outbound_mailbox_16.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_16.IRQS

*Table 1398. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_16.IRQEN

*Table 1399. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_16.IRQP

*Table 1400. axil_mailbox_smc_wrap.outbound_mailbox_16.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_16.CTRL

*Table 1401. axil_mailbox_smc_wrap.outbound_mailbox_16.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_16.WRITE_DATA

*Table 1402. axil_mailbox_smc_wrap.inbound_mailbox_16.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_16.READ_DATA

*Table 1403. axil_mailbox_smc_wrap.inbound_mailbox_16.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_16.STATUS

*Table 1404. axil_mailbox_smc_wrap.inbound_mailbox_16.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_16.ERROR_FLAGS

*Table 1405. axil_mailbox_smc_wrap.inbound_mailbox_16.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_16.WIRQT

*Table 1406. axil_mailbox_smc_wrap.inbound_mailbox_16.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_16.RIRQT

*Table 1407. axil_mailbox_smc_wrap.inbound_mailbox_16.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_16.IRQS

*Table 1408. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_16.IRQEN

*Table 1409. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_16.IRQP

*Table 1410. axil_mailbox_smc_wrap.inbound_mailbox_16.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_16.CTRL

*Table 1411. axil_mailbox_smc_wrap.inbound_mailbox_16.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_17.WRITE_DATA

*Table 1412. axil_mailbox_smc_wrap.outbound_mailbox_17.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_17.READ_DATA

*Table 1413. axil_mailbox_smc_wrap.outbound_mailbox_17.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_17.STATUS

*Table 1414. axil_mailbox_smc_wrap.outbound_mailbox_17.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_17.ERROR_FLAGS

*Table 1415. axil_mailbox_smc_wrap.outbound_mailbox_17.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_17.WIRQT

*Table 1416. axil_mailbox_smc_wrap.outbound_mailbox_17.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_17.RIRQT

*Table 1417. axil_mailbox_smc_wrap.outbound_mailbox_17.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_17.IRQS

*Table 1418. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_17.IRQEN

*Table 1419. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_17.IRQP

*Table 1420. axil_mailbox_smc_wrap.outbound_mailbox_17.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_17.CTRL

*Table 1421. axil_mailbox_smc_wrap.outbound_mailbox_17.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_17.WRITE_DATA

*Table 1422. axil_mailbox_smc_wrap.inbound_mailbox_17.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_17.READ_DATA

*Table 1423. axil_mailbox_smc_wrap.inbound_mailbox_17.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_17.STATUS

*Table 1424. axil_mailbox_smc_wrap.inbound_mailbox_17.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_17.ERROR_FLAGS

*Table 1425. axil_mailbox_smc_wrap.inbound_mailbox_17.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_17.WIRQT

*Table 1426. axil_mailbox_smc_wrap.inbound_mailbox_17.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_17.RIRQT

*Table 1427. axil_mailbox_smc_wrap.inbound_mailbox_17.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_17.IRQS

*Table 1428. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_17.IRQEN

*Table 1429. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_17.IRQP

*Table 1430. axil_mailbox_smc_wrap.inbound_mailbox_17.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_17.CTRL

*Table 1431. axil_mailbox_smc_wrap.inbound_mailbox_17.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_18.WRITE_DATA

*Table 1432. axil_mailbox_smc_wrap.outbound_mailbox_18.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_18.READ_DATA

*Table 1433. axil_mailbox_smc_wrap.outbound_mailbox_18.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_18.STATUS

*Table 1434. axil_mailbox_smc_wrap.outbound_mailbox_18.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_18.ERROR_FLAGS

*Table 1435. axil_mailbox_smc_wrap.outbound_mailbox_18.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_18.WIRQT

*Table 1436. axil_mailbox_smc_wrap.outbound_mailbox_18.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_18.RIRQT

*Table 1437. axil_mailbox_smc_wrap.outbound_mailbox_18.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_18.IRQS

*Table 1438. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_18.IRQEN

*Table 1439. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_18.IRQP

*Table 1440. axil_mailbox_smc_wrap.outbound_mailbox_18.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_18.CTRL

*Table 1441. axil_mailbox_smc_wrap.outbound_mailbox_18.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_18.WRITE_DATA

*Table 1442. axil_mailbox_smc_wrap.inbound_mailbox_18.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_18.READ_DATA

*Table 1443. axil_mailbox_smc_wrap.inbound_mailbox_18.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_18.STATUS

*Table 1444. axil_mailbox_smc_wrap.inbound_mailbox_18.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_18.ERROR_FLAGS

*Table 1445. axil_mailbox_smc_wrap.inbound_mailbox_18.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_18.WIRQT

*Table 1446. axil_mailbox_smc_wrap.inbound_mailbox_18.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_18.RIRQT

*Table 1447. axil_mailbox_smc_wrap.inbound_mailbox_18.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_18.IRQS

*Table 1448. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_18.IRQEN

*Table 1449. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_18.IRQP

*Table 1450. axil_mailbox_smc_wrap.inbound_mailbox_18.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_18.CTRL

*Table 1451. axil_mailbox_smc_wrap.inbound_mailbox_18.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_19.WRITE_DATA

*Table 1452. axil_mailbox_smc_wrap.outbound_mailbox_19.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_19.READ_DATA

*Table 1453. axil_mailbox_smc_wrap.outbound_mailbox_19.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_19.STATUS

*Table 1454. axil_mailbox_smc_wrap.outbound_mailbox_19.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_19.ERROR_FLAGS

*Table 1455. axil_mailbox_smc_wrap.outbound_mailbox_19.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_19.WIRQT

*Table 1456. axil_mailbox_smc_wrap.outbound_mailbox_19.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_19.RIRQT

*Table 1457. axil_mailbox_smc_wrap.outbound_mailbox_19.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_19.IRQS

*Table 1458. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_19.IRQEN

*Table 1459. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_19.IRQP

*Table 1460. axil_mailbox_smc_wrap.outbound_mailbox_19.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_19.CTRL

*Table 1461. axil_mailbox_smc_wrap.outbound_mailbox_19.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_19.WRITE_DATA

*Table 1462. axil_mailbox_smc_wrap.inbound_mailbox_19.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_19.READ_DATA

*Table 1463. axil_mailbox_smc_wrap.inbound_mailbox_19.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_19.STATUS

*Table 1464. axil_mailbox_smc_wrap.inbound_mailbox_19.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_19.ERROR_FLAGS

*Table 1465. axil_mailbox_smc_wrap.inbound_mailbox_19.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_19.WIRQT

*Table 1466. axil_mailbox_smc_wrap.inbound_mailbox_19.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_19.RIRQT

*Table 1467. axil_mailbox_smc_wrap.inbound_mailbox_19.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_19.IRQS

*Table 1468. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_19.IRQEN

*Table 1469. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_19.IRQP

*Table 1470. axil_mailbox_smc_wrap.inbound_mailbox_19.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_19.CTRL

*Table 1471. axil_mailbox_smc_wrap.inbound_mailbox_19.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_20.WRITE_DATA

*Table 1472. axil_mailbox_smc_wrap.outbound_mailbox_20.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_20.READ_DATA

*Table 1473. axil_mailbox_smc_wrap.outbound_mailbox_20.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_20.STATUS

*Table 1474. axil_mailbox_smc_wrap.outbound_mailbox_20.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_20.ERROR_FLAGS

*Table 1475. axil_mailbox_smc_wrap.outbound_mailbox_20.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_20.WIRQT

*Table 1476. axil_mailbox_smc_wrap.outbound_mailbox_20.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_20.RIRQT

*Table 1477. axil_mailbox_smc_wrap.outbound_mailbox_20.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_20.IRQS

*Table 1478. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_20.IRQEN

*Table 1479. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_20.IRQP

*Table 1480. axil_mailbox_smc_wrap.outbound_mailbox_20.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_20.CTRL

*Table 1481. axil_mailbox_smc_wrap.outbound_mailbox_20.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_20.WRITE_DATA

*Table 1482. axil_mailbox_smc_wrap.inbound_mailbox_20.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_20.READ_DATA

*Table 1483. axil_mailbox_smc_wrap.inbound_mailbox_20.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_20.STATUS

*Table 1484. axil_mailbox_smc_wrap.inbound_mailbox_20.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_20.ERROR_FLAGS

*Table 1485. axil_mailbox_smc_wrap.inbound_mailbox_20.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_20.WIRQT

*Table 1486. axil_mailbox_smc_wrap.inbound_mailbox_20.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_20.RIRQT

*Table 1487. axil_mailbox_smc_wrap.inbound_mailbox_20.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_20.IRQS

*Table 1488. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_20.IRQEN

*Table 1489. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_20.IRQP

*Table 1490. axil_mailbox_smc_wrap.inbound_mailbox_20.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_20.CTRL

*Table 1491. axil_mailbox_smc_wrap.inbound_mailbox_20.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_21.WRITE_DATA

*Table 1492. axil_mailbox_smc_wrap.outbound_mailbox_21.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_21.READ_DATA

*Table 1493. axil_mailbox_smc_wrap.outbound_mailbox_21.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_21.STATUS

*Table 1494. axil_mailbox_smc_wrap.outbound_mailbox_21.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_21.ERROR_FLAGS

*Table 1495. axil_mailbox_smc_wrap.outbound_mailbox_21.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_21.WIRQT

*Table 1496. axil_mailbox_smc_wrap.outbound_mailbox_21.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_21.RIRQT

*Table 1497. axil_mailbox_smc_wrap.outbound_mailbox_21.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_21.IRQS

*Table 1498. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_21.IRQEN

*Table 1499. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_21.IRQP

*Table 1500. axil_mailbox_smc_wrap.outbound_mailbox_21.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_21.CTRL

*Table 1501. axil_mailbox_smc_wrap.outbound_mailbox_21.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_21.WRITE_DATA

*Table 1502. axil_mailbox_smc_wrap.inbound_mailbox_21.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_21.READ_DATA

*Table 1503. axil_mailbox_smc_wrap.inbound_mailbox_21.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_21.STATUS

*Table 1504. axil_mailbox_smc_wrap.inbound_mailbox_21.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_21.ERROR_FLAGS

*Table 1505. axil_mailbox_smc_wrap.inbound_mailbox_21.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_21.WIRQT

*Table 1506. axil_mailbox_smc_wrap.inbound_mailbox_21.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_21.RIRQT

*Table 1507. axil_mailbox_smc_wrap.inbound_mailbox_21.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_21.IRQS

*Table 1508. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_21.IRQEN

*Table 1509. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_21.IRQP

*Table 1510. axil_mailbox_smc_wrap.inbound_mailbox_21.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_21.CTRL

*Table 1511. axil_mailbox_smc_wrap.inbound_mailbox_21.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_22.WRITE_DATA

*Table 1512. axil_mailbox_smc_wrap.outbound_mailbox_22.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_22.READ_DATA

*Table 1513. axil_mailbox_smc_wrap.outbound_mailbox_22.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_22.STATUS

*Table 1514. axil_mailbox_smc_wrap.outbound_mailbox_22.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_22.ERROR_FLAGS

*Table 1515. axil_mailbox_smc_wrap.outbound_mailbox_22.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_22.WIRQT

*Table 1516. axil_mailbox_smc_wrap.outbound_mailbox_22.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_22.RIRQT

*Table 1517. axil_mailbox_smc_wrap.outbound_mailbox_22.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_22.IRQS

*Table 1518. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_22.IRQEN

*Table 1519. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_22.IRQP

*Table 1520. axil_mailbox_smc_wrap.outbound_mailbox_22.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_22.CTRL

*Table 1521. axil_mailbox_smc_wrap.outbound_mailbox_22.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_22.WRITE_DATA

*Table 1522. axil_mailbox_smc_wrap.inbound_mailbox_22.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_22.READ_DATA

*Table 1523. axil_mailbox_smc_wrap.inbound_mailbox_22.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_22.STATUS

*Table 1524. axil_mailbox_smc_wrap.inbound_mailbox_22.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_22.ERROR_FLAGS

*Table 1525. axil_mailbox_smc_wrap.inbound_mailbox_22.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_22.WIRQT

*Table 1526. axil_mailbox_smc_wrap.inbound_mailbox_22.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_22.RIRQT

*Table 1527. axil_mailbox_smc_wrap.inbound_mailbox_22.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_22.IRQS

*Table 1528. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_22.IRQEN

*Table 1529. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_22.IRQP

*Table 1530. axil_mailbox_smc_wrap.inbound_mailbox_22.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_22.CTRL

*Table 1531. axil_mailbox_smc_wrap.inbound_mailbox_22.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_23.WRITE_DATA

*Table 1532. axil_mailbox_smc_wrap.outbound_mailbox_23.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_23.READ_DATA

*Table 1533. axil_mailbox_smc_wrap.outbound_mailbox_23.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_23.STATUS

*Table 1534. axil_mailbox_smc_wrap.outbound_mailbox_23.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_23.ERROR_FLAGS

*Table 1535. axil_mailbox_smc_wrap.outbound_mailbox_23.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_23.WIRQT

*Table 1536. axil_mailbox_smc_wrap.outbound_mailbox_23.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_23.RIRQT

*Table 1537. axil_mailbox_smc_wrap.outbound_mailbox_23.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_23.IRQS

*Table 1538. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_23.IRQEN

*Table 1539. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_23.IRQP

*Table 1540. axil_mailbox_smc_wrap.outbound_mailbox_23.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_23.CTRL

*Table 1541. axil_mailbox_smc_wrap.outbound_mailbox_23.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_23.WRITE_DATA

*Table 1542. axil_mailbox_smc_wrap.inbound_mailbox_23.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_23.READ_DATA

*Table 1543. axil_mailbox_smc_wrap.inbound_mailbox_23.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_23.STATUS

*Table 1544. axil_mailbox_smc_wrap.inbound_mailbox_23.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_23.ERROR_FLAGS

*Table 1545. axil_mailbox_smc_wrap.inbound_mailbox_23.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_23.WIRQT

*Table 1546. axil_mailbox_smc_wrap.inbound_mailbox_23.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_23.RIRQT

*Table 1547. axil_mailbox_smc_wrap.inbound_mailbox_23.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_23.IRQS

*Table 1548. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_23.IRQEN

*Table 1549. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_23.IRQP

*Table 1550. axil_mailbox_smc_wrap.inbound_mailbox_23.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_23.CTRL

*Table 1551. axil_mailbox_smc_wrap.inbound_mailbox_23.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_24.WRITE_DATA

*Table 1552. axil_mailbox_smc_wrap.outbound_mailbox_24.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_24.READ_DATA

*Table 1553. axil_mailbox_smc_wrap.outbound_mailbox_24.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_24.STATUS

*Table 1554. axil_mailbox_smc_wrap.outbound_mailbox_24.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_24.ERROR_FLAGS

*Table 1555. axil_mailbox_smc_wrap.outbound_mailbox_24.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_24.WIRQT

*Table 1556. axil_mailbox_smc_wrap.outbound_mailbox_24.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_24.RIRQT

*Table 1557. axil_mailbox_smc_wrap.outbound_mailbox_24.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_24.IRQS

*Table 1558. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_24.IRQEN

*Table 1559. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_24.IRQP

*Table 1560. axil_mailbox_smc_wrap.outbound_mailbox_24.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_24.CTRL

*Table 1561. axil_mailbox_smc_wrap.outbound_mailbox_24.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_24.WRITE_DATA

*Table 1562. axil_mailbox_smc_wrap.inbound_mailbox_24.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_24.READ_DATA

*Table 1563. axil_mailbox_smc_wrap.inbound_mailbox_24.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_24.STATUS

*Table 1564. axil_mailbox_smc_wrap.inbound_mailbox_24.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_24.ERROR_FLAGS

*Table 1565. axil_mailbox_smc_wrap.inbound_mailbox_24.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_24.WIRQT

*Table 1566. axil_mailbox_smc_wrap.inbound_mailbox_24.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_24.RIRQT

*Table 1567. axil_mailbox_smc_wrap.inbound_mailbox_24.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_24.IRQS

*Table 1568. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_24.IRQEN

*Table 1569. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_24.IRQP

*Table 1570. axil_mailbox_smc_wrap.inbound_mailbox_24.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_24.CTRL

*Table 1571. axil_mailbox_smc_wrap.inbound_mailbox_24.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_25.WRITE_DATA

*Table 1572. axil_mailbox_smc_wrap.outbound_mailbox_25.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_25.READ_DATA

*Table 1573. axil_mailbox_smc_wrap.outbound_mailbox_25.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_25.STATUS

*Table 1574. axil_mailbox_smc_wrap.outbound_mailbox_25.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_25.ERROR_FLAGS

*Table 1575. axil_mailbox_smc_wrap.outbound_mailbox_25.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_25.WIRQT

*Table 1576. axil_mailbox_smc_wrap.outbound_mailbox_25.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_25.RIRQT

*Table 1577. axil_mailbox_smc_wrap.outbound_mailbox_25.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_25.IRQS

*Table 1578. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_25.IRQEN

*Table 1579. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_25.IRQP

*Table 1580. axil_mailbox_smc_wrap.outbound_mailbox_25.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_25.CTRL

*Table 1581. axil_mailbox_smc_wrap.outbound_mailbox_25.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_25.WRITE_DATA

*Table 1582. axil_mailbox_smc_wrap.inbound_mailbox_25.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_25.READ_DATA

*Table 1583. axil_mailbox_smc_wrap.inbound_mailbox_25.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_25.STATUS

*Table 1584. axil_mailbox_smc_wrap.inbound_mailbox_25.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_25.ERROR_FLAGS

*Table 1585. axil_mailbox_smc_wrap.inbound_mailbox_25.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_25.WIRQT

*Table 1586. axil_mailbox_smc_wrap.inbound_mailbox_25.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_25.RIRQT

*Table 1587. axil_mailbox_smc_wrap.inbound_mailbox_25.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_25.IRQS

*Table 1588. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_25.IRQEN

*Table 1589. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_25.IRQP

*Table 1590. axil_mailbox_smc_wrap.inbound_mailbox_25.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_25.CTRL

*Table 1591. axil_mailbox_smc_wrap.inbound_mailbox_25.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_26.WRITE_DATA

*Table 1592. axil_mailbox_smc_wrap.outbound_mailbox_26.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_26.READ_DATA

*Table 1593. axil_mailbox_smc_wrap.outbound_mailbox_26.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_26.STATUS

*Table 1594. axil_mailbox_smc_wrap.outbound_mailbox_26.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_26.ERROR_FLAGS

*Table 1595. axil_mailbox_smc_wrap.outbound_mailbox_26.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_26.WIRQT

*Table 1596. axil_mailbox_smc_wrap.outbound_mailbox_26.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_26.RIRQT

*Table 1597. axil_mailbox_smc_wrap.outbound_mailbox_26.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_26.IRQS

*Table 1598. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_26.IRQEN

*Table 1599. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_26.IRQP

*Table 1600. axil_mailbox_smc_wrap.outbound_mailbox_26.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_26.CTRL

*Table 1601. axil_mailbox_smc_wrap.outbound_mailbox_26.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_26.WRITE_DATA

*Table 1602. axil_mailbox_smc_wrap.inbound_mailbox_26.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_26.READ_DATA

*Table 1603. axil_mailbox_smc_wrap.inbound_mailbox_26.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_26.STATUS

*Table 1604. axil_mailbox_smc_wrap.inbound_mailbox_26.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_26.ERROR_FLAGS

*Table 1605. axil_mailbox_smc_wrap.inbound_mailbox_26.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_26.WIRQT

*Table 1606. axil_mailbox_smc_wrap.inbound_mailbox_26.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_26.RIRQT

*Table 1607. axil_mailbox_smc_wrap.inbound_mailbox_26.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_26.IRQS

*Table 1608. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_26.IRQEN

*Table 1609. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_26.IRQP

*Table 1610. axil_mailbox_smc_wrap.inbound_mailbox_26.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_26.CTRL

*Table 1611. axil_mailbox_smc_wrap.inbound_mailbox_26.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_27.WRITE_DATA

*Table 1612. axil_mailbox_smc_wrap.outbound_mailbox_27.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_27.READ_DATA

*Table 1613. axil_mailbox_smc_wrap.outbound_mailbox_27.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_27.STATUS

*Table 1614. axil_mailbox_smc_wrap.outbound_mailbox_27.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_27.ERROR_FLAGS

*Table 1615. axil_mailbox_smc_wrap.outbound_mailbox_27.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_27.WIRQT

*Table 1616. axil_mailbox_smc_wrap.outbound_mailbox_27.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_27.RIRQT

*Table 1617. axil_mailbox_smc_wrap.outbound_mailbox_27.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_27.IRQS

*Table 1618. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_27.IRQEN

*Table 1619. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_27.IRQP

*Table 1620. axil_mailbox_smc_wrap.outbound_mailbox_27.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_27.CTRL

*Table 1621. axil_mailbox_smc_wrap.outbound_mailbox_27.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_27.WRITE_DATA

*Table 1622. axil_mailbox_smc_wrap.inbound_mailbox_27.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_27.READ_DATA

*Table 1623. axil_mailbox_smc_wrap.inbound_mailbox_27.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_27.STATUS

*Table 1624. axil_mailbox_smc_wrap.inbound_mailbox_27.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_27.ERROR_FLAGS

*Table 1625. axil_mailbox_smc_wrap.inbound_mailbox_27.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_27.WIRQT

*Table 1626. axil_mailbox_smc_wrap.inbound_mailbox_27.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_27.RIRQT

*Table 1627. axil_mailbox_smc_wrap.inbound_mailbox_27.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_27.IRQS

*Table 1628. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_27.IRQEN

*Table 1629. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_27.IRQP

*Table 1630. axil_mailbox_smc_wrap.inbound_mailbox_27.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_27.CTRL

*Table 1631. axil_mailbox_smc_wrap.inbound_mailbox_27.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_28.WRITE_DATA

*Table 1632. axil_mailbox_smc_wrap.outbound_mailbox_28.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_28.READ_DATA

*Table 1633. axil_mailbox_smc_wrap.outbound_mailbox_28.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_28.STATUS

*Table 1634. axil_mailbox_smc_wrap.outbound_mailbox_28.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_28.ERROR_FLAGS

*Table 1635. axil_mailbox_smc_wrap.outbound_mailbox_28.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_28.WIRQT

*Table 1636. axil_mailbox_smc_wrap.outbound_mailbox_28.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_28.RIRQT

*Table 1637. axil_mailbox_smc_wrap.outbound_mailbox_28.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_28.IRQS

*Table 1638. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_28.IRQEN

*Table 1639. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_28.IRQP

*Table 1640. axil_mailbox_smc_wrap.outbound_mailbox_28.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_28.CTRL

*Table 1641. axil_mailbox_smc_wrap.outbound_mailbox_28.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_28.WRITE_DATA

*Table 1642. axil_mailbox_smc_wrap.inbound_mailbox_28.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_28.READ_DATA

*Table 1643. axil_mailbox_smc_wrap.inbound_mailbox_28.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_28.STATUS

*Table 1644. axil_mailbox_smc_wrap.inbound_mailbox_28.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_28.ERROR_FLAGS

*Table 1645. axil_mailbox_smc_wrap.inbound_mailbox_28.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_28.WIRQT

*Table 1646. axil_mailbox_smc_wrap.inbound_mailbox_28.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_28.RIRQT

*Table 1647. axil_mailbox_smc_wrap.inbound_mailbox_28.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_28.IRQS

*Table 1648. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_28.IRQEN

*Table 1649. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_28.IRQP

*Table 1650. axil_mailbox_smc_wrap.inbound_mailbox_28.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_28.CTRL

*Table 1651. axil_mailbox_smc_wrap.inbound_mailbox_28.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_29.WRITE_DATA

*Table 1652. axil_mailbox_smc_wrap.outbound_mailbox_29.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_29.READ_DATA

*Table 1653. axil_mailbox_smc_wrap.outbound_mailbox_29.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_29.STATUS

*Table 1654. axil_mailbox_smc_wrap.outbound_mailbox_29.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_29.ERROR_FLAGS

*Table 1655. axil_mailbox_smc_wrap.outbound_mailbox_29.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_29.WIRQT

*Table 1656. axil_mailbox_smc_wrap.outbound_mailbox_29.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_29.RIRQT

*Table 1657. axil_mailbox_smc_wrap.outbound_mailbox_29.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_29.IRQS

*Table 1658. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_29.IRQEN

*Table 1659. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_29.IRQP

*Table 1660. axil_mailbox_smc_wrap.outbound_mailbox_29.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_29.CTRL

*Table 1661. axil_mailbox_smc_wrap.outbound_mailbox_29.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_29.WRITE_DATA

*Table 1662. axil_mailbox_smc_wrap.inbound_mailbox_29.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_29.READ_DATA

*Table 1663. axil_mailbox_smc_wrap.inbound_mailbox_29.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_29.STATUS

*Table 1664. axil_mailbox_smc_wrap.inbound_mailbox_29.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_29.ERROR_FLAGS

*Table 1665. axil_mailbox_smc_wrap.inbound_mailbox_29.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_29.WIRQT

*Table 1666. axil_mailbox_smc_wrap.inbound_mailbox_29.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_29.RIRQT

*Table 1667. axil_mailbox_smc_wrap.inbound_mailbox_29.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_29.IRQS

*Table 1668. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_29.IRQEN

*Table 1669. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_29.IRQP

*Table 1670. axil_mailbox_smc_wrap.inbound_mailbox_29.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_29.CTRL

*Table 1671. axil_mailbox_smc_wrap.inbound_mailbox_29.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_30.WRITE_DATA

*Table 1672. axil_mailbox_smc_wrap.outbound_mailbox_30.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_30.READ_DATA

*Table 1673. axil_mailbox_smc_wrap.outbound_mailbox_30.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_30.STATUS

*Table 1674. axil_mailbox_smc_wrap.outbound_mailbox_30.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_30.ERROR_FLAGS

*Table 1675. axil_mailbox_smc_wrap.outbound_mailbox_30.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_30.WIRQT

*Table 1676. axil_mailbox_smc_wrap.outbound_mailbox_30.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_30.RIRQT

*Table 1677. axil_mailbox_smc_wrap.outbound_mailbox_30.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_30.IRQS

*Table 1678. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_30.IRQEN

*Table 1679. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_30.IRQP

*Table 1680. axil_mailbox_smc_wrap.outbound_mailbox_30.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_30.CTRL

*Table 1681. axil_mailbox_smc_wrap.outbound_mailbox_30.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_30.WRITE_DATA

*Table 1682. axil_mailbox_smc_wrap.inbound_mailbox_30.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_30.READ_DATA

*Table 1683. axil_mailbox_smc_wrap.inbound_mailbox_30.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_30.STATUS

*Table 1684. axil_mailbox_smc_wrap.inbound_mailbox_30.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_30.ERROR_FLAGS

*Table 1685. axil_mailbox_smc_wrap.inbound_mailbox_30.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_30.WIRQT

*Table 1686. axil_mailbox_smc_wrap.inbound_mailbox_30.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_30.RIRQT

*Table 1687. axil_mailbox_smc_wrap.inbound_mailbox_30.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_30.IRQS

*Table 1688. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_30.IRQEN

*Table 1689. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_30.IRQP

*Table 1690. axil_mailbox_smc_wrap.inbound_mailbox_30.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_30.CTRL

*Table 1691. axil_mailbox_smc_wrap.inbound_mailbox_30.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_31.WRITE_DATA

*Table 1692. axil_mailbox_smc_wrap.outbound_mailbox_31.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_31.READ_DATA

*Table 1693. axil_mailbox_smc_wrap.outbound_mailbox_31.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_31.STATUS

*Table 1694. axil_mailbox_smc_wrap.outbound_mailbox_31.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_31.ERROR_FLAGS

*Table 1695. axil_mailbox_smc_wrap.outbound_mailbox_31.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_31.WIRQT

*Table 1696. axil_mailbox_smc_wrap.outbound_mailbox_31.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_31.RIRQT

*Table 1697. axil_mailbox_smc_wrap.outbound_mailbox_31.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_31.IRQS

*Table 1698. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_31.IRQEN

*Table 1699. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_31.IRQP

*Table 1700. axil_mailbox_smc_wrap.outbound_mailbox_31.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_31.CTRL

*Table 1701. axil_mailbox_smc_wrap.outbound_mailbox_31.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_31.WRITE_DATA

*Table 1702. axil_mailbox_smc_wrap.inbound_mailbox_31.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_31.READ_DATA

*Table 1703. axil_mailbox_smc_wrap.inbound_mailbox_31.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_31.STATUS

*Table 1704. axil_mailbox_smc_wrap.inbound_mailbox_31.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_31.ERROR_FLAGS

*Table 1705. axil_mailbox_smc_wrap.inbound_mailbox_31.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_31.WIRQT

*Table 1706. axil_mailbox_smc_wrap.inbound_mailbox_31.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_31.RIRQT

*Table 1707. axil_mailbox_smc_wrap.inbound_mailbox_31.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_31.IRQS

*Table 1708. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_31.IRQEN

*Table 1709. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_31.IRQP

*Table 1710. axil_mailbox_smc_wrap.inbound_mailbox_31.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_31.CTRL

*Table 1711. axil_mailbox_smc_wrap.inbound_mailbox_31.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### SEP AXI-Lite Mailbox Wrapper Address Map

SEP-side wrapper instantiating the AXI-Lite mailbox pairs. Each outbound_mailbox_N / inbound_mailbox_N pair is the two apertures of one bidirectional mailbox and shares a single pair of FIFOs: data written through WRITE_DATA in one aperture is what READ_DATA in the other aperture returns. The outbound/inbound naming reflects intended use, not an access restriction.

*Table 1712. SEP AXI-Lite Mailbox Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | outbound_mailbox_0.WRITE_DATA | W | - |
| 0x8 | outbound_mailbox_0.READ_DATA | R | - |
| 0x10 | outbound_mailbox_0.STATUS | R | - |
| 0x18 | outbound_mailbox_0.ERROR_FLAGS | R | - |
| 0x20 | outbound_mailbox_0.WIRQT | RW | - |
| 0x28 | outbound_mailbox_0.RIRQT | RW | - |
| 0x30 | outbound_mailbox_0.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x38 | outbound_mailbox_0.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x40 | outbound_mailbox_0.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x48 | outbound_mailbox_0.CTRL | W | - |
| 0x800 | inbound_mailbox_0.WRITE_DATA | W | - |
| 0x808 | inbound_mailbox_0.READ_DATA | R | - |
| 0x810 | inbound_mailbox_0.STATUS | R | - |
| 0x818 | inbound_mailbox_0.ERROR_FLAGS | R | - |
| 0x820 | inbound_mailbox_0.WIRQT | RW | - |
| 0x828 | inbound_mailbox_0.RIRQT | RW | - |
| 0x830 | inbound_mailbox_0.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x838 | inbound_mailbox_0.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x840 | inbound_mailbox_0.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x848 | inbound_mailbox_0.CTRL | W | - |
| 0x1000 | outbound_mailbox_1.WRITE_DATA | W | - |
| 0x1008 | outbound_mailbox_1.READ_DATA | R | - |
| 0x1010 | outbound_mailbox_1.STATUS | R | - |
| 0x1018 | outbound_mailbox_1.ERROR_FLAGS | R | - |
| 0x1020 | outbound_mailbox_1.WIRQT | RW | - |
| 0x1028 | outbound_mailbox_1.RIRQT | RW | - |
| 0x1030 | outbound_mailbox_1.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1038 | outbound_mailbox_1.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1040 | outbound_mailbox_1.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1048 | outbound_mailbox_1.CTRL | W | - |
| 0x1800 | inbound_mailbox_1.WRITE_DATA | W | - |
| 0x1808 | inbound_mailbox_1.READ_DATA | R | - |
| 0x1810 | inbound_mailbox_1.STATUS | R | - |
| 0x1818 | inbound_mailbox_1.ERROR_FLAGS | R | - |
| 0x1820 | inbound_mailbox_1.WIRQT | RW | - |
| 0x1828 | inbound_mailbox_1.RIRQT | RW | - |
| 0x1830 | inbound_mailbox_1.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x1838 | inbound_mailbox_1.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x1840 | inbound_mailbox_1.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x1848 | inbound_mailbox_1.CTRL | W | - |
| 0x2000 | outbound_mailbox_2.WRITE_DATA | W | - |
| 0x2008 | outbound_mailbox_2.READ_DATA | R | - |
| 0x2010 | outbound_mailbox_2.STATUS | R | - |
| 0x2018 | outbound_mailbox_2.ERROR_FLAGS | R | - |
| 0x2020 | outbound_mailbox_2.WIRQT | RW | - |
| 0x2028 | outbound_mailbox_2.RIRQT | RW | - |
| 0x2030 | outbound_mailbox_2.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x2038 | outbound_mailbox_2.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x2040 | outbound_mailbox_2.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x2048 | outbound_mailbox_2.CTRL | W | - |
| 0x2800 | inbound_mailbox_2.WRITE_DATA | W | - |
| 0x2808 | inbound_mailbox_2.READ_DATA | R | - |
| 0x2810 | inbound_mailbox_2.STATUS | R | - |
| 0x2818 | inbound_mailbox_2.ERROR_FLAGS | R | - |
| 0x2820 | inbound_mailbox_2.WIRQT | RW | - |
| 0x2828 | inbound_mailbox_2.RIRQT | RW | - |
| 0x2830 | inbound_mailbox_2.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x2838 | inbound_mailbox_2.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x2840 | inbound_mailbox_2.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x2848 | inbound_mailbox_2.CTRL | W | - |
| 0x3000 | outbound_mailbox_3.WRITE_DATA | W | - |
| 0x3008 | outbound_mailbox_3.READ_DATA | R | - |
| 0x3010 | outbound_mailbox_3.STATUS | R | - |
| 0x3018 | outbound_mailbox_3.ERROR_FLAGS | R | - |
| 0x3020 | outbound_mailbox_3.WIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x3028 | outbound_mailbox_3.RIRQT | RW | - |
| 0x3030 | outbound_mailbox_3.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x3038 | outbound_mailbox_3.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x3040 | outbound_mailbox_3.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x3048 | outbound_mailbox_3.CTRL | W | - |
| 0x3800 | inbound_mailbox_3.WRITE_DATA | W | - |
| 0x3808 | inbound_mailbox_3.READ_DATA | R | - |
| 0x3810 | inbound_mailbox_3.STATUS | R | - |
| 0x3818 | inbound_mailbox_3.ERROR_FLAGS | R | - |
| 0x3820 | inbound_mailbox_3.WIRQT | RW | - |
| 0x3828 | inbound_mailbox_3.RIRQT | RW | - |
| 0x3830 | inbound_mailbox_3.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x3838 | inbound_mailbox_3.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x3840 | inbound_mailbox_3.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x3848 | inbound_mailbox_3.CTRL | W | - |
| 0x4000 | outbound_mailbox_4.WRITE_DATA | W | - |
| 0x4008 | outbound_mailbox_4.READ_DATA | R | - |
| 0x4010 | outbound_mailbox_4.STATUS | R | - |
| 0x4018 | outbound_mailbox_4.ERROR_FLAGS | R | - |
| 0x4020 | outbound_mailbox_4.WIRQT | RW | - |
| 0x4028 | outbound_mailbox_4.RIRQT | RW | - |
| 0x4030 | outbound_mailbox_4.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x4038 | outbound_mailbox_4.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x4040 | outbound_mailbox_4.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x4048 | outbound_mailbox_4.CTRL | W | - |
| 0x4800 | inbound_mailbox_4.WRITE_DATA | W | - |
| 0x4808 | inbound_mailbox_4.READ_DATA | R | - |
| 0x4810 | inbound_mailbox_4.STATUS | R | - |
| 0x4818 | inbound_mailbox_4.ERROR_FLAGS | R | - |
| 0x4820 | inbound_mailbox_4.WIRQT | RW | - |
| 0x4828 | inbound_mailbox_4.RIRQT | RW | - |
| 0x4830 | inbound_mailbox_4.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x4838 | inbound_mailbox_4.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x4840 | inbound_mailbox_4.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x4848 | inbound_mailbox_4.CTRL | W | - |
| 0x5000 | outbound_mailbox_5.WRITE_DATA | W | - |
| 0x5008 | outbound_mailbox_5.READ_DATA | R | - |
| 0x5010 | outbound_mailbox_5.STATUS | R | - |
| 0x5018 | outbound_mailbox_5.ERROR_FLAGS | R | - |
| 0x5020 | outbound_mailbox_5.WIRQT | RW | - |
| 0x5028 | outbound_mailbox_5.RIRQT | RW | - |
| 0x5030 | outbound_mailbox_5.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x5038 | outbound_mailbox_5.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x5040 | outbound_mailbox_5.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x5048 | outbound_mailbox_5.CTRL | W | - |
| 0x5800 | inbound_mailbox_5.WRITE_DATA | W | - |
| 0x5808 | inbound_mailbox_5.READ_DATA | R | - |
| 0x5810 | inbound_mailbox_5.STATUS | R | - |
| 0x5818 | inbound_mailbox_5.ERROR_FLAGS | R | - |
| 0x5820 | inbound_mailbox_5.WIRQT | RW | - |
| 0x5828 | inbound_mailbox_5.RIRQT | RW | - |
| 0x5830 | inbound_mailbox_5.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x5838 | inbound_mailbox_5.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x5840 | inbound_mailbox_5.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x5848 | inbound_mailbox_5.CTRL | W | - |
| 0x6000 | outbound_mailbox_6.WRITE_DATA | W | - |
| 0x6008 | outbound_mailbox_6.READ_DATA | R | - |
| 0x6010 | outbound_mailbox_6.STATUS | R | - |
| 0x6018 | outbound_mailbox_6.ERROR_FLAGS | R | - |
| 0x6020 | outbound_mailbox_6.WIRQT | RW | - |
| 0x6028 | outbound_mailbox_6.RIRQT | RW | - |
| 0x6030 | outbound_mailbox_6.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x6038 | outbound_mailbox_6.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x6040 | outbound_mailbox_6.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x6048 | outbound_mailbox_6.CTRL | W | - |
| 0x6800 | inbound_mailbox_6.WRITE_DATA | W | - |
| 0x6808 | inbound_mailbox_6.READ_DATA | R | - |
| 0x6810 | inbound_mailbox_6.STATUS | R | - |
| 0x6818 | inbound_mailbox_6.ERROR_FLAGS | R | - |
| 0x6820 | inbound_mailbox_6.WIRQT | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x6828 | inbound_mailbox_6.RIRQT | RW | - |
| 0x6830 | inbound_mailbox_6.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x6838 | inbound_mailbox_6.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x6840 | inbound_mailbox_6.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x6848 | inbound_mailbox_6.CTRL | W | - |
| 0x7000 | outbound_mailbox_7.WRITE_DATA | W | - |
| 0x7008 | outbound_mailbox_7.READ_DATA | R | - |
| 0x7010 | outbound_mailbox_7.STATUS | R | - |
| 0x7018 | outbound_mailbox_7.ERROR_FLAGS | R | - |
| 0x7020 | outbound_mailbox_7.WIRQT | RW | - |
| 0x7028 | outbound_mailbox_7.RIRQT | RW | - |
| 0x7030 | outbound_mailbox_7.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x7038 | outbound_mailbox_7.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x7040 | outbound_mailbox_7.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x7048 | outbound_mailbox_7.CTRL | W | - |
| 0x7800 | inbound_mailbox_7.WRITE_DATA | W | - |
| 0x7808 | inbound_mailbox_7.READ_DATA | R | - |
| 0x7810 | inbound_mailbox_7.STATUS | R | - |
| 0x7818 | inbound_mailbox_7.ERROR_FLAGS | R | - |
| 0x7820 | inbound_mailbox_7.WIRQT | RW | - |
| 0x7828 | inbound_mailbox_7.RIRQT | RW | - |
| 0x7830 | inbound_mailbox_7.IRQS | RW | This register is used to read and clear interrupt requests, regardless of IRQEN register and whether the interrupt is enabled or disabled. |
| 0x7838 | inbound_mailbox_7.IRQEN | RW | This register is used to enable and disable which interrupts are sent to the CPU. |
| 0x7840 | inbound_mailbox_7.IRQP | R | This register is used to read the current status of the interrupts sent to the CPU. |
| 0x7848 | inbound_mailbox_7.CTRL | W | - |

### outbound_mailbox_0.WRITE_DATA

*Table 1713. axil_mailbox_sep_wrap.outbound_mailbox_0.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_0.READ_DATA

*Table 1714. axil_mailbox_sep_wrap.outbound_mailbox_0.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_0.STATUS

*Table 1715. axil_mailbox_sep_wrap.outbound_mailbox_0.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_0.ERROR_FLAGS

*Table 1716. axil_mailbox_sep_wrap.outbound_mailbox_0.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_0.WIRQT

*Table 1717. axil_mailbox_sep_wrap.outbound_mailbox_0.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_0.RIRQT

*Table 1718. axil_mailbox_sep_wrap.outbound_mailbox_0.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_0.IRQS

*Table 1719. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_0.IRQEN

*Table 1720. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_0.IRQP

*Table 1721. axil_mailbox_sep_wrap.outbound_mailbox_0.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_0.CTRL

*Table 1722. axil_mailbox_sep_wrap.outbound_mailbox_0.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_0.WRITE_DATA

*Table 1723. axil_mailbox_sep_wrap.inbound_mailbox_0.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_0.READ_DATA

*Table 1724. axil_mailbox_sep_wrap.inbound_mailbox_0.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_0.STATUS

*Table 1725. axil_mailbox_sep_wrap.inbound_mailbox_0.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_0.ERROR_FLAGS

*Table 1726. axil_mailbox_sep_wrap.inbound_mailbox_0.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_0.WIRQT

*Table 1727. axil_mailbox_sep_wrap.inbound_mailbox_0.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_0.RIRQT

*Table 1728. axil_mailbox_sep_wrap.inbound_mailbox_0.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_0.IRQS

*Table 1729. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_0.IRQEN

*Table 1730. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_0.IRQP

*Table 1731. axil_mailbox_sep_wrap.inbound_mailbox_0.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_0.CTRL

*Table 1732. axil_mailbox_sep_wrap.inbound_mailbox_0.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_1.WRITE_DATA

*Table 1733. axil_mailbox_sep_wrap.outbound_mailbox_1.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_1.READ_DATA

*Table 1734. axil_mailbox_sep_wrap.outbound_mailbox_1.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_1.STATUS

*Table 1735. axil_mailbox_sep_wrap.outbound_mailbox_1.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_1.ERROR_FLAGS

*Table 1736. axil_mailbox_sep_wrap.outbound_mailbox_1.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_1.WIRQT

*Table 1737. axil_mailbox_sep_wrap.outbound_mailbox_1.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_1.RIRQT

*Table 1738. axil_mailbox_sep_wrap.outbound_mailbox_1.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_1.IRQS

*Table 1739. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_1.IRQEN

*Table 1740. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_1.IRQP

*Table 1741. axil_mailbox_sep_wrap.outbound_mailbox_1.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_1.CTRL

*Table 1742. axil_mailbox_sep_wrap.outbound_mailbox_1.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_1.WRITE_DATA

*Table 1743. axil_mailbox_sep_wrap.inbound_mailbox_1.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_1.READ_DATA

*Table 1744. axil_mailbox_sep_wrap.inbound_mailbox_1.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_1.STATUS

*Table 1745. axil_mailbox_sep_wrap.inbound_mailbox_1.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_1.ERROR_FLAGS

*Table 1746. axil_mailbox_sep_wrap.inbound_mailbox_1.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_1.WIRQT

*Table 1747. axil_mailbox_sep_wrap.inbound_mailbox_1.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_1.RIRQT

*Table 1748. axil_mailbox_sep_wrap.inbound_mailbox_1.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_1.IRQS

*Table 1749. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_1.IRQEN

*Table 1750. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_1.IRQP

*Table 1751. axil_mailbox_sep_wrap.inbound_mailbox_1.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_1.CTRL

*Table 1752. axil_mailbox_sep_wrap.inbound_mailbox_1.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_2.WRITE_DATA

*Table 1753. axil_mailbox_sep_wrap.outbound_mailbox_2.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_2.READ_DATA

*Table 1754. axil_mailbox_sep_wrap.outbound_mailbox_2.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_2.STATUS

*Table 1755. axil_mailbox_sep_wrap.outbound_mailbox_2.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_2.ERROR_FLAGS

*Table 1756. axil_mailbox_sep_wrap.outbound_mailbox_2.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_2.WIRQT

*Table 1757. axil_mailbox_sep_wrap.outbound_mailbox_2.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_2.RIRQT

*Table 1758. axil_mailbox_sep_wrap.outbound_mailbox_2.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_2.IRQS

*Table 1759. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_2.IRQEN

*Table 1760. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_2.IRQP

*Table 1761. axil_mailbox_sep_wrap.outbound_mailbox_2.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_2.CTRL

*Table 1762. axil_mailbox_sep_wrap.outbound_mailbox_2.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_2.WRITE_DATA

*Table 1763. axil_mailbox_sep_wrap.inbound_mailbox_2.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_2.READ_DATA

*Table 1764. axil_mailbox_sep_wrap.inbound_mailbox_2.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_2.STATUS

*Table 1765. axil_mailbox_sep_wrap.inbound_mailbox_2.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_2.ERROR_FLAGS

*Table 1766. axil_mailbox_sep_wrap.inbound_mailbox_2.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_2.WIRQT

*Table 1767. axil_mailbox_sep_wrap.inbound_mailbox_2.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_2.RIRQT

*Table 1768. axil_mailbox_sep_wrap.inbound_mailbox_2.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_2.IRQS

*Table 1769. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_2.IRQEN

*Table 1770. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_2.IRQP

*Table 1771. axil_mailbox_sep_wrap.inbound_mailbox_2.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_2.CTRL

*Table 1772. axil_mailbox_sep_wrap.inbound_mailbox_2.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_3.WRITE_DATA

*Table 1773. axil_mailbox_sep_wrap.outbound_mailbox_3.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_3.READ_DATA

*Table 1774. axil_mailbox_sep_wrap.outbound_mailbox_3.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_3.STATUS

*Table 1775. axil_mailbox_sep_wrap.outbound_mailbox_3.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_3.ERROR_FLAGS

*Table 1776. axil_mailbox_sep_wrap.outbound_mailbox_3.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_3.WIRQT

*Table 1777. axil_mailbox_sep_wrap.outbound_mailbox_3.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_3.RIRQT

*Table 1778. axil_mailbox_sep_wrap.outbound_mailbox_3.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_3.IRQS

*Table 1779. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_3.IRQEN

*Table 1780. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_3.IRQP

*Table 1781. axil_mailbox_sep_wrap.outbound_mailbox_3.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_3.CTRL

*Table 1782. axil_mailbox_sep_wrap.outbound_mailbox_3.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_3.WRITE_DATA

*Table 1783. axil_mailbox_sep_wrap.inbound_mailbox_3.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_3.READ_DATA

*Table 1784. axil_mailbox_sep_wrap.inbound_mailbox_3.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_3.STATUS

*Table 1785. axil_mailbox_sep_wrap.inbound_mailbox_3.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_3.ERROR_FLAGS

*Table 1786. axil_mailbox_sep_wrap.inbound_mailbox_3.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_3.WIRQT

*Table 1787. axil_mailbox_sep_wrap.inbound_mailbox_3.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_3.RIRQT

*Table 1788. axil_mailbox_sep_wrap.inbound_mailbox_3.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_3.IRQS

*Table 1789. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_3.IRQEN

*Table 1790. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_3.IRQP

*Table 1791. axil_mailbox_sep_wrap.inbound_mailbox_3.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_3.CTRL

*Table 1792. axil_mailbox_sep_wrap.inbound_mailbox_3.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_4.WRITE_DATA

*Table 1793. axil_mailbox_sep_wrap.outbound_mailbox_4.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_4.READ_DATA

*Table 1794. axil_mailbox_sep_wrap.outbound_mailbox_4.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_4.STATUS

*Table 1795. axil_mailbox_sep_wrap.outbound_mailbox_4.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_4.ERROR_FLAGS

*Table 1796. axil_mailbox_sep_wrap.outbound_mailbox_4.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_4.WIRQT

*Table 1797. axil_mailbox_sep_wrap.outbound_mailbox_4.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_4.RIRQT

*Table 1798. axil_mailbox_sep_wrap.outbound_mailbox_4.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_4.IRQS

*Table 1799. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_4.IRQEN

*Table 1800. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_4.IRQP

*Table 1801. axil_mailbox_sep_wrap.outbound_mailbox_4.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_4.CTRL

*Table 1802. axil_mailbox_sep_wrap.outbound_mailbox_4.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_4.WRITE_DATA

*Table 1803. axil_mailbox_sep_wrap.inbound_mailbox_4.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_4.READ_DATA

*Table 1804. axil_mailbox_sep_wrap.inbound_mailbox_4.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_4.STATUS

*Table 1805. axil_mailbox_sep_wrap.inbound_mailbox_4.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_4.ERROR_FLAGS

*Table 1806. axil_mailbox_sep_wrap.inbound_mailbox_4.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_4.WIRQT

*Table 1807. axil_mailbox_sep_wrap.inbound_mailbox_4.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_4.RIRQT

*Table 1808. axil_mailbox_sep_wrap.inbound_mailbox_4.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_4.IRQS

*Table 1809. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_4.IRQEN

*Table 1810. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_4.IRQP

*Table 1811. axil_mailbox_sep_wrap.inbound_mailbox_4.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_4.CTRL

*Table 1812. axil_mailbox_sep_wrap.inbound_mailbox_4.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_5.WRITE_DATA

*Table 1813. axil_mailbox_sep_wrap.outbound_mailbox_5.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_5.READ_DATA

*Table 1814. axil_mailbox_sep_wrap.outbound_mailbox_5.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_5.STATUS

*Table 1815. axil_mailbox_sep_wrap.outbound_mailbox_5.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_5.ERROR_FLAGS

*Table 1816. axil_mailbox_sep_wrap.outbound_mailbox_5.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_5.WIRQT

*Table 1817. axil_mailbox_sep_wrap.outbound_mailbox_5.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_5.RIRQT

*Table 1818. axil_mailbox_sep_wrap.outbound_mailbox_5.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_5.IRQS

*Table 1819. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_5.IRQEN

*Table 1820. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_5.IRQP

*Table 1821. axil_mailbox_sep_wrap.outbound_mailbox_5.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_5.CTRL

*Table 1822. axil_mailbox_sep_wrap.outbound_mailbox_5.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_5.WRITE_DATA

*Table 1823. axil_mailbox_sep_wrap.inbound_mailbox_5.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_5.READ_DATA

*Table 1824. axil_mailbox_sep_wrap.inbound_mailbox_5.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_5.STATUS

*Table 1825. axil_mailbox_sep_wrap.inbound_mailbox_5.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_5.ERROR_FLAGS

*Table 1826. axil_mailbox_sep_wrap.inbound_mailbox_5.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_5.WIRQT

*Table 1827. axil_mailbox_sep_wrap.inbound_mailbox_5.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_5.RIRQT

*Table 1828. axil_mailbox_sep_wrap.inbound_mailbox_5.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_5.IRQS

*Table 1829. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_5.IRQEN

*Table 1830. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_5.IRQP

*Table 1831. axil_mailbox_sep_wrap.inbound_mailbox_5.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_5.CTRL

*Table 1832. axil_mailbox_sep_wrap.inbound_mailbox_5.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_6.WRITE_DATA

*Table 1833. axil_mailbox_sep_wrap.outbound_mailbox_6.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_6.READ_DATA

*Table 1834. axil_mailbox_sep_wrap.outbound_mailbox_6.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_6.STATUS

*Table 1835. axil_mailbox_sep_wrap.outbound_mailbox_6.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_6.ERROR_FLAGS

*Table 1836. axil_mailbox_sep_wrap.outbound_mailbox_6.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_6.WIRQT

*Table 1837. axil_mailbox_sep_wrap.outbound_mailbox_6.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_6.RIRQT

*Table 1838. axil_mailbox_sep_wrap.outbound_mailbox_6.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_6.IRQS

*Table 1839. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_6.IRQEN

*Table 1840. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_6.IRQP

*Table 1841. axil_mailbox_sep_wrap.outbound_mailbox_6.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_6.CTRL

*Table 1842. axil_mailbox_sep_wrap.outbound_mailbox_6.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_6.WRITE_DATA

*Table 1843. axil_mailbox_sep_wrap.inbound_mailbox_6.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_6.READ_DATA

*Table 1844. axil_mailbox_sep_wrap.inbound_mailbox_6.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_6.STATUS

*Table 1845. axil_mailbox_sep_wrap.inbound_mailbox_6.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_6.ERROR_FLAGS

*Table 1846. axil_mailbox_sep_wrap.inbound_mailbox_6.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_6.WIRQT

*Table 1847. axil_mailbox_sep_wrap.inbound_mailbox_6.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_6.RIRQT

*Table 1848. axil_mailbox_sep_wrap.inbound_mailbox_6.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_6.IRQS

*Table 1849. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_6.IRQEN

*Table 1850. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_6.IRQP

*Table 1851. axil_mailbox_sep_wrap.inbound_mailbox_6.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### inbound_mailbox_6.CTRL

*Table 1852. axil_mailbox_sep_wrap.inbound_mailbox_6.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### outbound_mailbox_7.WRITE_DATA

*Table 1853. axil_mailbox_sep_wrap.outbound_mailbox_7.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### outbound_mailbox_7.READ_DATA

*Table 1854. axil_mailbox_sep_wrap.outbound_mailbox_7.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### outbound_mailbox_7.STATUS

*Table 1855. axil_mailbox_sep_wrap.outbound_mailbox_7.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### outbound_mailbox_7.ERROR_FLAGS

*Table 1856. axil_mailbox_sep_wrap.outbound_mailbox_7.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### outbound_mailbox_7.WIRQT

*Table 1857. axil_mailbox_sep_wrap.outbound_mailbox_7.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### outbound_mailbox_7.RIRQT

*Table 1858. axil_mailbox_sep_wrap.outbound_mailbox_7.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### outbound_mailbox_7.IRQS

*Table 1859. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### outbound_mailbox_7.IRQEN

*Table 1860. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### outbound_mailbox_7.IRQP

*Table 1861. axil_mailbox_sep_wrap.outbound_mailbox_7.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

### outbound_mailbox_7.CTRL

*Table 1862. axil_mailbox_sep_wrap.outbound_mailbox_7.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### inbound_mailbox_7.WRITE_DATA

*Table 1863. axil_mailbox_sep_wrap.inbound_mailbox_7.WRITE_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `write_data` | W WMOD | 0x0 | Write data to mailbox fifo |

### inbound_mailbox_7.READ_DATA

*Table 1864. axil_mailbox_sep_wrap.inbound_mailbox_7.READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `read_data` | R RMOD | - | Read data from mailbox fifo |

### inbound_mailbox_7.STATUS

*Table 1865. axil_mailbox_sep_wrap.inbound_mailbox_7.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:4 | `Reserved` | - | - | Reserved |
| 3 | `read_level_above_thresh` | R | 0x0 | If set, read fifo level is higher than threshold set in RIRQT |
| 2 | `write_level_above_thresh` | R | 0x0 | If set, write fifo level is higher than threshold set in WIRQT |
| 1 | `full` | R | 0x0 | 0: Space is available to write, 1: Space is not available to write |
| 0 | `empty` | R | 0x1 | 0: Data is available to read, 1: Data is not available to read |

### inbound_mailbox_7.ERROR_FLAGS

*Table 1866. axil_mailbox_sep_wrap.inbound_mailbox_7.ERROR_FLAGS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `write_error` | R RC | 0x0 | Attempted write to a full mailbox |
| 0 | `read_error` | R RC | 0x0 | Attempted read from an empty mailbox |

### inbound_mailbox_7.WIRQT

*Table 1867. axil_mailbox_sep_wrap.inbound_mailbox_7.WIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `wirqt` | RW | 0x0 | When the usage pointer of the FIFO connected to the W channel exceeds this value, a write threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the write FIFO is full. |

### inbound_mailbox_7.RIRQT

*Table 1868. axil_mailbox_sep_wrap.inbound_mailbox_7.RIRQT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `rirqt` | RW | 0x0 | When the fill pointer of the FIFO connected to the R channel exceeds this value, a read threshold IRQ is triggered and the corresponding STATUS register bit is set. When a value larger than or equal to the MailboxDepth parameter is written to this register, it gets reduced to MailboxDepth - 1 to ensure an IRQ is triggered when the read FIFO is full. |

### inbound_mailbox_7.IRQS

*Table 1869. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Error on mailbox access On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 1 | `rtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in read mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |
| 0 | `wtirq` | RW1C | 0x0 | On read: [0]: No interrupt request [1]: Usage level threshold in write mailbox exceeded On write: [0]: No acknowledge [1]: Acknowledge and clear interrupt request |

### inbound_mailbox_7.IRQEN

*Table 1870. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQEN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | RW | 0x0 | [0]: Error IRQ disabled [1]: Error IRQ enabled |
| 1 | `rtirq` | RW | 0x0 | [0]: Read threshold IRQ disabled [1]: Read threshold IRQ enabled |
| 0 | `wtirq` | RW | 0x0 | [0]: Write threshold IRQ disabled [1]: Write threshold IRQ enabled |

### inbound_mailbox_7.IRQP

*Table 1871. axil_mailbox_sep_wrap.inbound_mailbox_7.IRQP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:3 | `Reserved` | - | - | Reserved |
| 2 | `eirq` | R | 0x0 | [0]: No error IRQ pending [1]: Error IRQ pending |
| 1 | `rtirq` | R | 0x0 | [0]: No read threshold IRQ pending [1]: Read threshold IRQ pending |
| 0 | `wtirq` | R | 0x0 | [0]: No write threshold IRQ pending [1]: Write threshold IRQ pending |

#### inbound_mailbox_7.CTRL

*Table 1872. axil_mailbox_sep_wrap.inbound_mailbox_7.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:2 | `Reserved` | - | - | Reserved |
| 1 | `rflush` | W WMOD | 0x0 | Flush the read FIFO for this port |
| 0 | `wflush` | W WMOD | 0x0 | Flush the write FIFO for this port |

### System Timer OCTS

#### Overview

The System Timer implements Open Chiplet Time Synchronization (OCTS). A primary timer counts local clock cycles and sends load and credit pulses to secondary timers. Each secondary advances between pulses, then corrects its count at the next credit boundary.

The counter is 64 bits. Its time unit depends on the primary clock; resolution and alignment accuracy also depend on the clock ratios, pulse routing and synchronization delay. The block does not guarantee nanosecond accuracy or simultaneous sampling across chiplets.

Software configures the timer through a 32-bit AXI4-Lite interface. The `is_primary_i` input selects its role. The sections below cover counting, synchronization constraints, setup and register access.

#### Operation

Figure 65 shows how primary and secondary timers exchange synchronization information. The dashed groups hold the logic that acts in only one mode, as `is_primary_i` selects it; the enable and the 64-bit count serve both. The inbound sync-load and credit pulses cross into `clk_i` through a flop and a `prim_edge_detector`, which synchronizes them with two flops and turns each rising edge into a one-cycle pulse; an AND gate with `is_primary_i` on its inverted input masks each pulse in PRIMARY mode. The enable flop is sticky: an OR gate holds it set once a PRIMARY-mode `TIMER_START.START` or a sync-load pulse has set it. In SECONDARY mode, the sync-load pulse loads the preset, a credit pulse moves the count to the expected count plus CREDIT_VAL, and between pulses the count advances by `CTRL.STEP`, which drives the core's `timer_cnt_step_i`, while credits are left. The credit tracking block holds the `cur_credits` accumulator, which adds STEP each cycle while it is below CREDIT_VAL (`credits_left`); the expected count, which `u_adder_expected_credit` advances by CREDIT_VAL on each credit pulse; and the `credit_expired_o` counter of cycles spent without credits. Both inbound pulses reset `cur_credits` and `credit_expired_o`. In PRIMARY mode, the count advances by one, and the credit counter and pulse state machine drive the outbound pulses. The wrapper keeps the longest credit-starved run reported by the core in a flop that loads the larger of its own value and `credit_expired_o`, and reads it back as `CREDIT_EXPIRED`; a write to `CREDIT_EXPIRED` clears both. In the SMC, the timer pulses use GPIO pads 55 and 56, which drive them on the primary chiplet and sample them on a secondary one.

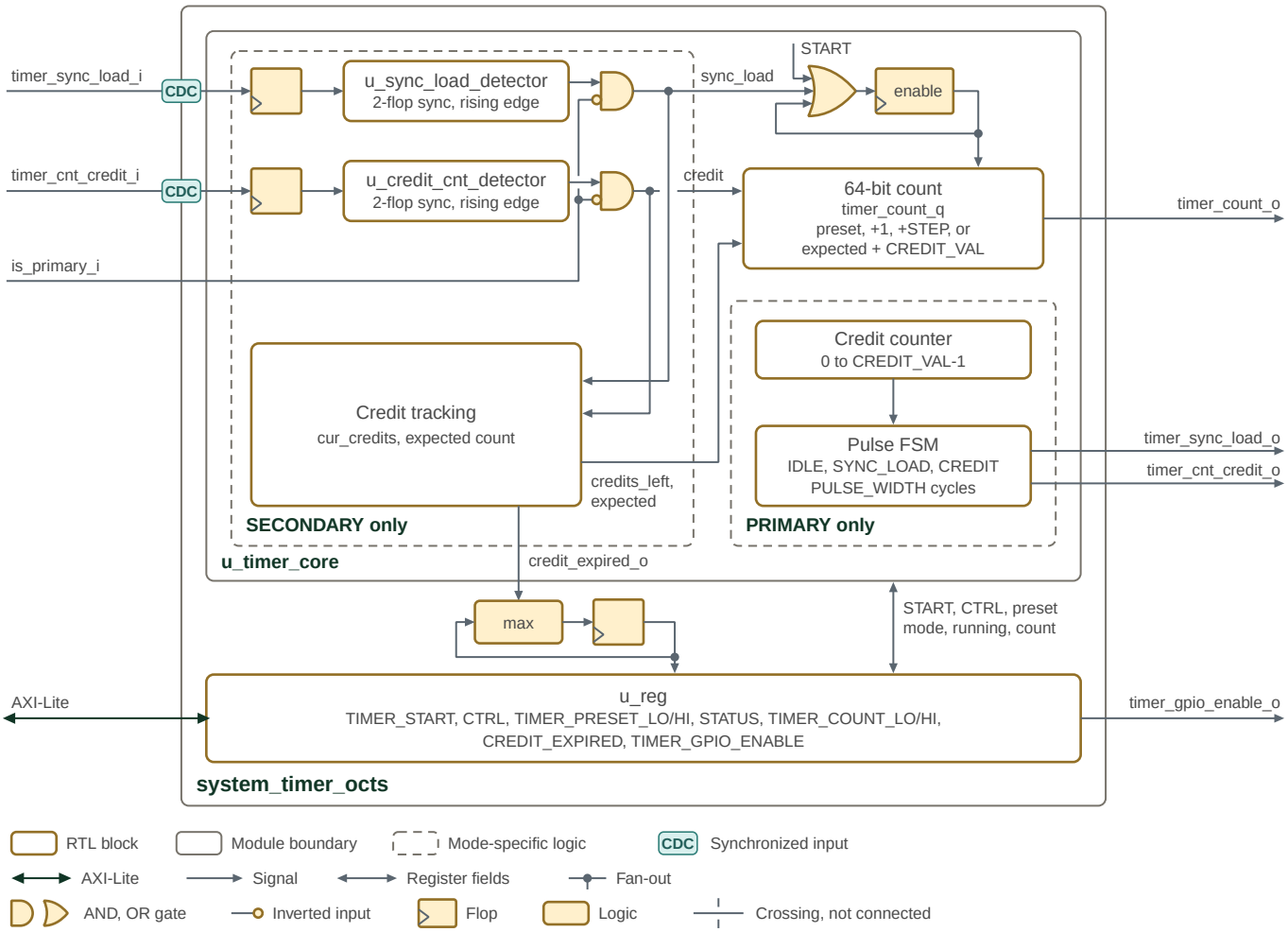


*Figure 65. System Timer OCTS primary/secondary synchronization and credit CDC*

### Counting and Synchronization

A write of 1 to `TIMER_START.START` starts the primary and loads `TIMER_PRESET_HI:LO`. The primary then increments by one on each `clk_i` cycle. It ignores synchronization inputs and generates load and credit pulses for the secondaries.

A secondary loads its own preset when it detects a rising edge on `timer_sync_load_i`. On each credit edge, it advances its expected count by `CTRL.CREDIT_VAL` and replaces its actual count with that expected value. Between edges, it increments by `CTRL.STEP` while credits remain. This replacement is a correction, not just a diagnostic comparison: a secondary count can move backward if its previous increments overshot the boundary.

*Table 1873. OCTS credit management*

| Role | Behavior |
|---|---|
| Primary | An 8-bit interval counter generates a credit event every `CREDIT_VAL` cycles while enabled. START clears this interval counter. |
| Secondary | A 9-bit consumed-credit counter starts at zero on load or credit edges. While it is less than `CREDIT_VAL`, both it and the timer advance by `STEP`. Counting pauses once the credit limit is reached. |

Choose a nonzero secondary `STEP` appropriate to the clock ratio. The limit is checked before adding the step, so the last increment can cross it. A lost credit pulse also leaves the expected count behind the primary; receiving a later credit does not recover the missing interval.

### Pulse Generation and Clock Crossing

*Table 1874. OCTS pulse states*

| State | Behavior |
|---|---|
| `PULSE_IDLE` | Accept START, or a credit event while enabled. START has priority. |
| `PULSE_SYNC_LOAD` | Assert the primary load output for the configured width. |
| `PULSE_CREDIT` | Assert the primary credit output for the configured width. |

`CTRL.PULSE_WIDTH` sets the high duration in primary clock cycles; zero is treated as one. `CREDIT_VAL` must exceed this effective width. The FSM accepts triggers only while idle and does not queue them. In particular, START can reload the primary count while an active pulse prevents a new load pulse; use a coordinated reset and setup sequence for restarting.

Each synchronization input passes through an input flop, a two-flop synchronizer and an edge detector in the receiving clock domain. Pulses must stay high long enough to be sampled and low long enough to distinguish successive events. Determine those intervals from the destination clock, routing delay and CDC analysis; the RTL defines no universal frequency ratio, nanosecond timing guarantee or fixed system-wide alignment bound.

### Status and Credit Expiration

`STATUS.MODE` is 0 for primary and 1 for secondary. `STATUS.RUNNING` is set when the internal enable is set and the count is nonzero. It can therefore remain set while a secondary is paused for lack of credit, or remain clear immediately after loading a zero preset. It is not a synchronization check.

The secondary core counts cycles spent without credit. Load and credit edges clear this interval counter. The register wrapper retains the largest observed value in `CREDIT_EXPIRED`; writing any value clears the saved maximum, while only a load or credit pulse clears the running interval counter in the core. Normal credit reception does not clear the saved maximum. In a primary configured from reset, it stays zero.

Use this value to detect waiting for credit, with thresholds derived from the clock ratio and pulse interval. It does not directly measure timestamp error or prove that all credit pulses arrived.

### Reset

`rst_ni` asynchronously clears the counter, enable, pulse state and registers. Reset release must meet the integration's timing requirements; this block does not synchronize reset deassertion internally.

Enable is sticky after a PRIMARY-mode START or a received load edge. There is no stop CSR; writing START=0 does not stop counting. Configure role and timing before starting, and use system reset when a coordinated reconfiguration is needed.

### Interface

The System Timer OCTS IP has a 32-bit AXI4-Lite register interface. The `is_primary_i` input selects PRIMARY or SECONDARY mode at runtime. All signals are synchronous to `clk_i` except `timer_sync_load_i` and `timer_cnt_credit_i`, which are synchronized internally as described in Pulse Generation and Clock Crossing.

### Ports

The ports of `system_timer_octs` are listed in the HTML edition of this manual, under RTL Modules Reference.

`system_timer_octs` has no parameters. Its AXI4-Lite request and response types, with 32-bit address and data, are defined

in `system_timer_octs_pkg`.

### OCTS Synchronization Interface

When operating in PRIMARY mode, the timer generates synchronization pulses: `timer_sync_load_o` is asserted for `CTRL.PULSE_WIDTH` cycles when the START register is written, and `timer_cnt_credit_o` is asserted for `CTRL.PULSE_WIDTH` cycles every `CTRL.CREDIT_VAL` cycles. Any synchronization input signals are ignored in PRIMARY mode. In contrast, SECONDARY mode relies on input from the PRIMARY: `timer_sync_load_i` directs the timer to initialize using a preset value, and `timer_cnt_credit_i` delivers credit pulses that allow continuous synchronization. In this mode, the outputs remain inactive.

### Timer Count

`timer_count_o` is the live 64-bit count, also readable through `TIMER_COUNT_LO` and `TIMER_COUNT_HI`. A PRIMARY advances it by one each `clk_i` cycle. A SECONDARY advances it by `CTRL.STEP` each cycle while credits remain.

### Timing

The RTL defines no setup, hold or clock-to-output limit, maximum clock frequency or frequency ratio between OCTS clock domains. Clock frequency, jitter, reset timing and physical timing constraints must come from the integration timing specification.

Synchronization inputs cross into `clk_i` as described in Pulse Generation and Clock Crossing; derive the required pulse high and low intervals from the destination clock and CDC analysis. Reset behavior is described in Reset.

## Memory Map and Register Reference

### Memory Map

The register bank occupies `0x00`–`0x23` and uses 32-bit AXI4-Lite accesses. Use the enclosing subsystem's memory map for its base address.

*Table 1875. OCTS register offsets and reset values*

| Register | Offset | Reset | Meaning |
|---|---|---|---|
| `TIMER_START` | `0x00` | 0 | START is a single-cycle command. |
| `CTRL` | `0x04` | `0x0001020A` | STEP=1, PULSE_WIDTH=2, CREDIT_VAL=10. |
| `STATUS` | `0x08` | 0 | MODE subsequently reflects the role input; RUNNING reflects enable and nonzero count. |
| `TIMER_PRESET_LO/HI` | `0x0C/0x10` | 0 | Two independently writable preset halves. |
| `TIMER_COUNT_LO/HI` | `0x14/0x18` | 0 | Live count halves, with no snapshot latch; the timestamp read sequence is in the Programmer's Guide. |
| `CREDIT_EXPIRED` | `0x1C` | 0 | Saved maximum wait for credit; any write clears it. |
| `TIMER_GPIO_ENABLE` | `0x20` | 0 | Timer GPIO LSIO selection starts disabled. |

## Detailed Register Map

### OCTS System Timer Address Map

Open Chiplet Time Synchronization (OCTS) System Timer Registers

*Table 1876. OCTS System Timer Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | TIMER_START | RW | System Timer Start Register |
| 0x4 | CTRL | RW | System Timer Control Register |
| 0x8 | STATUS | R | System Timer Status Register |
| 0xC | TIMER_PRESET_LO | RW | System Timer Preset Value [31:0] |
| 0x10 | TIMER_PRESET_HI | RW | System Timer Preset Value [63:32] |
| 0x14 | TIMER_COUNT_LO | R | Current System Timer Count [31:0] |
| 0x18 | TIMER_COUNT_HI | R | Current System Timer Count [63:32] |
| 0x1C | CREDIT_EXPIRED | RW | Credit Expired Register |
| 0x20 | TIMER_GPIO_ENABLE | RW | System Timer GPIO Enable Register |

### TIMER_START

*Table 1877. system_timer_octs.TIMER_START fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `START` | RW 1P | 0x0 | Start the system timer. In PRIMARY mode, a write of 1 loads the preset and pulses the sync-load output for PULSE_WIDTH cycles. In SECONDARY mode, the write has no effect; a secondary starts on the primary's sync-load pulse, which is what loads its preset. Ensure both primary and secondary are out of reset. |

### CTRL

*Table 1878. system_timer_octs.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:16 | `STEP` | RW | 0x1 | Step amount for secondary timer |
| 15:8 | `PULSE_WIDTH` | RW | 0x2 | Pulse width for sync load and credit signals. A value of 0 will be rounded up to 1. WARNING: This value must be less than CREDIT_VAL |
| 7:0 | `CREDIT_VAL` | RW | 0xA | Credit value factor. WARNING: This value must be greater than PULSE_WIDTH |

### STATUS

*Table 1879. system_timer_octs.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `RUNNING` | R | 0x0 | Timer is enabled and its count is nonzero. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MODE` | R | 0x0 | Timer mode driven by the is_primary_i input: 0=PRIMARY (is_primary_i=1), 1=SECONDARY (is_primary_i=0). |

### TIMER_PRESET_LO

*Table 1880. system_timer_octs.TIMER_PRESET_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PRESET_LO | RW | 0x0 | Lower 32 bits of preset value |

### TIMER_PRESET_HI

*Table 1881. system_timer_octs.TIMER_PRESET_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PRESET_HI | RW | 0x0 | Upper 32 bits of preset value |

### TIMER_COUNT_LO

*Table 1882. system_timer_octs.TIMER_COUNT_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | COUNT_LO | R | 0x0 | Lower 32 bits of current timer count |

### TIMER_COUNT_HI

*Table 1883. system_timer_octs.TIMER_COUNT_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | COUNT_HI | R | 0x0 | Upper 32 bits of current timer count |

### CREDIT_EXPIRED

*Table 1884. system_timer_octs.CREDIT_EXPIRED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | MAX_CYCLES_EXPIRED | RW | 0x0 | Maximum number of clock cycles since a count credit expired (SECONDARY only). To reset the value of this register, write anything to it. If this number is very high (exact value depends on clock speed and credit value), this indicates that the secondary has not received a count credit pulse in a while, and is likely not syncing properly with the primary. This register can also be used to determine the clock skew between the primary and secondary. If PRIMARY, this field is always 0. |

### TIMER_GPIO_ENABLE

*Table 1885. system_timer_octs.TIMER_GPIO_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | - | - | Reserved |
| 0 | GPIO_ENABLE | RW | 0x0 | Enable the GPIO pad lsio interface for the system timer. This is done to prevent X-prop on reset into the secondary timer, which can cause the secondary timer to start counting early. |

## eFuse

**NOTE** In this chapter, **eFuse** and **OTP** are used interchangeably.

### Overview

The eFuse IP provides secure One-Time Programmable (OTP) memory access for the OCAH system. A foundry-agnostic controller caches the array into shadow registers at boot, then exposes program and read operations over AXI4-Lite with layered hardware and software locks. Foundry-specific timing and macro signaling sit behind a replaceable SHIM so the software interface stays the same across OTP technologies.

### Description and Features

- **Boot-time fuse sense**: An automatic bulk read loads the full OTP array into shadow registers so later CSR access is not bound to OTP timing.
- **Dual AXI4-Lite paths**: SMC is the functional access path; JTAG is the debug path, with lifecycle-dependent restrictions.
- **Lifecycle and token gating**: SEP instances further restrict access by lifecycle state, secure test mode, and RMA or security-disable tokens.

### Architecture

The eFuse IP provides secure One-Time Programmable (OTP) memory management with foundry-agnostic design. The layered architecture separates security control logic from foundry-specific implementation details, enabling deployment across different OTP technologies while maintaining consistent security characteristics and software interfaces.

### Interface Controller

The eFuse Interface Controller (efuse_interface_controller.sv) provides dual-path access control with security policy enforcement.

### Access Paths

- **SMC AXI4-Lite Interface**: Functional access for normal system operations
- **JTAG AXI4-Lite Interface**: Debug access for manufacturing test and development

### Command-path ownership

After fuse sense completes, software program and read requests share one command path to the OTP bank. `EFUSE_PROGRAM_CTRL.program_enable` takes that path in preference to `EFUSE_READ_CTRL.read_enable`, so the two enables must be asserted exclusively; a read issued while `program_enable` is set does not reach the bank. The program and read sequences are in the eFuse section of the Programmer's Guide.

### Destination Endpoints

Table 1886 identifies the consumers of sensed fuse data.

*Table 1886. eFuse destination endpoints*

| Endpoint | Description |
|---|---|
| Shadow Registers | High-performance boot-time caching for rapid access without OTP timing constraints |
| eFuse CSR | Control and status registers for read, program, and verification operations |
| MMR | Memory-Mapped Registers for SEP security tokens with specialized access control |
| Interface SHIM CSR | Configuration registers for foundry-specific timing adaptation. Implemented by the adopter, but always accessed via the interface controller. |

### Shadow Register Subsystem

The shadow register subsystem accelerates OTP access by caching eFuse contents at boot time. On power-up, a dedicated FSM automatically issues a bulk read command (with `access_length_words` set to cover the full eFuse bank), copying all OTP data into the shadow registers. This ensures the shadow registers are fully populated and always synchronized with the OTP state at each boot.

### Features

Table 1887 summarises the shadow-register capacity and access features.

*Table 1887. eFuse shadow-register features*

| Feature | Description |
|---|---|
| Capacity | Configurable (default: 8192 bits / 1KB, 256 × 32-bit words), set via the `SHADOW_REG_BITS` parameter |
| Initialization | Automatic boot-time loading via Fuse Sense FSM |
| Access Control | Multi-tier hardware and software lock mechanisms |
| Data Consistency | Maintains synchronization with underlying OTP storage upon each power-on reset |

### State Machine Architecture

The read interface FSM (efuse_read_interface.sv) operates in a simple two-state sequence, transitioning from idle to waiting for a response as it handles on-demand eFuse reads that fall outside the shadow register cache. This mechanism works alongside the shadow subsystem, providing direct access only when necessary, and actively reports status to support robust error handling and retry logic.

The program interface FSM (efuse_program_interface.sv) enforces the write-once nature of OTP hardware, ensuring that programmed data cannot be altered once set. It supports write-and-verify procedures for reliability, integrating verification steps that check programmed values against intended data, and fully blocks any unauthorized overwrites to maintain data integrity.

**Readback Capture Behavior:**

When a read command completes successfully, the read interface captures data from the SHIM response into the read-data CSR during the ST_WAIT_RESP state. This data is retained and continues to be available after the FSM returns to idle, allowing software to retrieve the result without a re-read.

### Parameterized Command Typedefs

The fuse command request/response structs exchanged between the controller and the SHIM are not declared with fixed types. Instead, they are generated from parameterized preprocessor macros defined in `efuse_typedef.svh`. This keeps the command interface technology-agnostic: each integration supplies its own address, data, access-length, and command types, while the macros guarantee a consistent struct layout and field ordering across all OTP variants.

Table 1888 relates each command macro to its generated type and purpose.

*Table 1888. eFuse command type macros*

| Macro | Generated Type | Purpose |
|---|---|---|
| `EFUSE_COMMAND_REQ_T` | `fuse_command_req_t` | Controller-to-SHIM command request |
| `EFUSE_COMMAND_RESP_T` | `fuse_command_resp_t` | SHIM-to-controller command response |

The request macro is parameterized over the struct name plus four payload types (`addr_t`, `data_t`, `access_length_words_t`,

command_t), allowing the controller to size the address space, transfer length, and command encoding to match the target OTP technology:

```
`define EFUSE_COMMAND_REQ_T(fuse_command_req_t, addr_t, data_t, access_length_words_t,
command_t)  \
  typedef struct packed {                                        \
    addr_t                    address;             // Target address       \
    logic                     program_data;        // Data to program (1'b1) \
    access_length_words_t     access_length_words; // Transfer size        \
    command_t                 command;             // Command type         \
    logic                     valid;               // Request valid        \
  } fuse_command_req_t;
```

The response macro is parameterized over the struct name and the data_t read-data type:

```
`define EFUSE_COMMAND_RESP_T(fuse_command_resp_t, data_t)       \
  typedef struct packed {                                       \
    data_t              data;       // Read data                \
    logic               status;     // Operation status         \
    logic               valid;      // Response valid            \
  } fuse_command_resp_t;
```

A typical instantiation expands both macros with the integration's local types:

```
`EFUSE_COMMAND_REQ_T(fuse_command_req_t, addr_t, data_t, access_length_words_t,
command_t)
`EFUSE_COMMAND_RESP_T(fuse_command_resp_t, data_t)
```

#### Foundry-Specific Signals

Table 1889 compares the signals required by the supported OTP interface variants.

*Table 1889. Example Hardware Interfaces by OTP Variant*

| OTP Variant | Hardware Interface Signals |
|---|---|
| Adopter OTP macro | Adopter-defined macro-side signals |
| Current Example Model | APB interface with register-based control |

#### Interface SHIM Layer

The Interface SHIM (efuse_interface_shim.sv) is an example, foundry-specific implementation of the custom fuse command interface. Its role is to translate the controller's generic commands into the signaling expected by a particular OTP macro; in this reference version the macro-side interface is APB, but a real integration would replace this with the foundry's native signals.

The SHIM decodes three command types from the controller:

- **Read** — fetch one or more words from the bank.
- **Program** — burn new bits into the bank.
- **Program/read-back** — program, then read back to verify success.

It implements these with dedicated read and program FSMs that sequence the macro-side transactions and return a `fuse_command_resp_t` to the controller. Foundry-specific timing and configuration (for example, program pulse widths) are exposed through the SHIM Control CSRs, accessed over the `fuse_bank_ctrl` AXI4-Lite interface.

#### eFuse Bank Model

The eFuse bank model (`efuse_bank_model.sv`) is a reference, simulation-only stand-in for the real foundry OTP macro. It is a 3KB (768 x 32-bit word) store fronted by a standard APB interface, and it emulates the set-once behavior of real OTP so the controller and SHIM can be exercised end-to-end without silicon. In a production integration it is replaced by the actual foundry macro driven by the SHIM.

#### Access Permissions and Security

Access control is layered for defense in depth: per-field permissions are described by `efuse_field_map_i`, lock state is enforced by the eFuse guard (`efuse_guard.sv`), security-disable tokens are validated, and access can additionally be gated by the chip lifecycle state.

Each eFuse field is described by a `rule_t` entry that pairs an address range with a 4-bit `lock` permission field:

```
typedef struct packed {
    logic [EfuseFieldMapIdxWidth-1:0] idx;        // Field index
    logic [3:0]                           lock;       // Permission bits
    logic [31:0]                          start_addr; // Field start address
    logic [31:0]                          end_addr;   // Field end address
} rule_t;
```

The `lock` field encodes read, write, and secure-test-mode permissions for the field (Table 1890).

*Table 1890. eFuse lock-field encoding*

| Bits | Function | Encoding |
|---|---|---|
| `lock[0]` | Read | `0` = readable, `1` = read-locked |
| `lock[2:1]` | Write | `00` = unlocked, `10` = set-only (1's may be added, never cleared), `11` = write-locked |
| `lock[3]` | Secure-test write lock | `1` adds a shadow write lock while secure test mode is active; `0` adds no such lock. Other locks still apply. |

#### Hardware vs. Software Locks

The same permission semantics are reached through two independent mechanisms, and the guard enforces whichever is more restrictive (Table 1891).

*Table 1891. eFuse hardware and software locks*

| Property | Hardware Lock | Software Lock |
|---|---|---|
| Source | Permission bits fused into the eFuse array (the field map reflects the programmed values) | Runtime writes to the `LOCKS` CSR (write-1-to-set) |

| Property | Hardware Lock | Software Lock |
|---|---|---|
| Persistence | Permanent — backed by one-time-programmable bits and cannot be cleared once set | Volatile — retained only until the next system reset |
| Reversibility | Irreversible by any means | Cleared by reset, allowing the field to be re-opened |
| Typical use | Final production lockdown of provisioned secrets | Temporary protection during boot, debug, or staged provisioning |

Because software locks are write-1-to-set, firmware can tighten access at runtime but can never loosen a restriction without a reset; hardware locks cannot be loosened at all. On top of these per-field locks, sensitive fields can be further gated by the chip lifecycle state and by secure test mode (`lock[3]`), so a field may be accessible in one operational mode and fully blocked in another even when its read/write bits are clear.

For sensitive functions, such as those governed by the MMR registers, the system enforces token-based security. Only by providing the correct 256-bit tokens, such as the RMA SOP Token for state transitions, the RMA Chiplet Token for chiplet-specific operations, or the Security Disable Token to override protections, can access to these resources be granted.

#### Security Disable Bypass Mechanism

The eFuse interface controller supports a security disable bypass mechanism via the `security_disable_i` input signal. This mechanism is primarily used for test and debug scenarios where the automatic fuse sensing sequence needs to be bypassed.

**Behavior when `security_disable_i` is asserted:**

Table 1892 describes the sensing behavior when security_disable_i is asserted.

*Table 1892. eFuse security-disable behavior*

| Behavior | Description |
|---|---|
| FSM Bypass | The automatic fuse sensing state machine is bypassed and does not issue read commands to the physical eFuse bank. |
| Shadow Register Output | Shadow register values are output directly to downstream logic via `shadow_regs_o`, regardless of `fuse_sense_done` status. |
| Shadow Register Loading | Shadow registers are not loaded from fuse command responses; they retain their reset values (all zeros for SMC). |
| APB Access | APB access to shadow registers is permitted without waiting for `fuse_sense_done` completion. |
| Reset Values | When the system is in reset state with `security_disable_i` asserted, shadow registers are initialized to zero (default reset value for SMC). |

For SMC, the internal security disable signal is directly driven by the `security_disable_i` input port.

#### Lifecycle State (LC_STATE) Effects

The eFuse interface controller enforces access restrictions based on the chip's lifecycle state.

**Lifecycle State Encoding:**

Table 1893 lists the lifecycle encodings used by the eFuse controller.

*Table 1893. eFuse lifecycle encodings*

| State | Encoding | Description |
|---|---|---|
| TEST_DEV | 0x0 | Test/Development state (default) |

| State | Encoding | Description |
|---|---|---|
| PROD | 0x1 | Production state |
| PROD_DBG | 0x1 + DEMOTE=1 | Production Debug (volatile, clears on reset) |
| RMA_SOP | 0x2, 0x3 | RMA System-in-Package |
| RMA_CHIPLET | 0x6-0x7 | RMA Chiplet level |
| PROD_END | 0x8 | Production End (terminal state) |
| INVALID | All others | Invalid state (error fallback) |

**JTAG Access Control Based on LC_STATE:**

JTAG access policies differ between SMC and SEP variants. The table below describes the SMC variant policy, which restricts JTAG access in production and RMA states. SEP applies a different policy: in restricted states (PROD or RMA_SIP), JTAG accesses to the MMR (token register) space remain allowed for token programming; accesses outside the MMR space are blocked with an error response.

**SMC JTAG Policy:**

When lifecycle state is PROD (0x1) or RMA_SOP (0x2, 0x3), JTAG access to eFuse registers is restricted (Table 1894).

*Table 1894. eFuse JTAG permissions (SMC)*

| Access Type | PROD/RMA_SOP State | Other States |
|---|---|---|
| JTAG Write | **Blocked** (error response) | Allowed |
| JTAG Read | **Blocked** except JTAG_PUBLIC_IDENTITY | Allowed |
| JTAG Read JTAG_PUBLIC_IDENTITY | Allowed | Allowed |

The access control mechanism uses an AXI-Lite demux that routes blocked requests to an error slave. When a request is blocked, the error slave returns an error response with data value `0xbadcab1e`.

**SMC exception for JTAG_PUBLIC_IDENTITY:**

Even in PROD or RMA_SOP states, JTAG read access to the JTAG_PUBLIC_IDENTITY register is permitted. This allows identification of the chip for manufacturing and field returns without exposing sensitive eFuse contents.

**Secure Test Mode (secure_tm_i)**

The eFuse interface controller supports a Secure Test Mode via the `secure_tm_i` input signal. This mode provides additional protection for sensitive security assets during secure manufacturing test operations.

**Behavior when `secure_tm_i` is asserted:**

Table 1895 describes the restrictions applied when secure_tm_i is asserted.

*Table 1895. eFuse secure-test behavior*

| Behavior | Description |
|---|---|
| Fuse Command Blocking | All fuse commands to the physical eFuse bank are blocked. The guard logic filters incoming requests and returns default responses. |
| RMA Token Protection | RMA SOP and RMA Chiplet tokens are disconnected from shadow register outputs (set to zero), preventing token exposure during test. |
| Write Lock Contribution | Secure test mode contributes to write lock status, preventing shadow register modifications. |

| Behavior | Description |
|---|---|
| Response Behavior | When fuse commands are blocked, default response values (all zeros) are returned to requesters. |

**SMC Configuration:**

For SMC, `secure_tm_i` is tied to logic `0`, meaning Secure Test Mode is not active during normal SMC operation. This signal is primarily relevant for SEP where sensitive RMA tokens must be protected during manufacturing test flows.

### Interface

#### Configuration Parameters

The OTP has a total storage capacity of 1 KB, organized as 256 words of 32 bits.

The parameters and ports of `efuse_interface_controller` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Main AXI4-Lite Interfaces

The eFuse controller provides two AXI4-Lite interfaces for register access: functional access from the SMC and debug access from JTAG.

#### Fuse Bank Control Interface

AXI4-Lite interface to the fuse bank control registers in the SHIM, carried on `fuse_bank_ctrl_req_o` and `fuse_bank_ctrl_resp_i` of types `efuse_axil_req_t` and `efuse_axil_resp_t`. The controller forwards accesses outside its own address windows on this interface.

#### Fuse Command Interface

Custom interface for SHIM state machine communication.

#### Security and Control Signals

The controller has no lifecycle-state input port. When `HAS_LC_STATE` is set, it takes the lifecycle state from the fuses: during fuse sensing it loads the low `LC_STATE_WIDTH` bits of shadow word `efuse_pkg::ShadowIdxLcState` and stores them differentially encoded in the shadow map. The shadow word index is fixed in `efuse_pkg`. `LC_STATE_BIT_POSITION` is used only by the guard, which derives the RMA SiP and chiplet token bit addresses from it as `LC_STATE_BIT_POSITION + 1` and `LC_STATE_BIT_POSITION + 2`.

#### Fuse Command Protocol

Custom command interface between controller and SHIM:

**Request Structure** (`fuse_command_req_t`):

```
typedef struct packed {
    addr_t                    address;           // Target address in bits
    logic                     program_data;      // Data to program (1'b1)
    access_length_words_t     access_length_words; // Transfer size
    command_t                 command;           // Command type
```

```
    logic                    valid;            // Request valid
} fuse_command_req_t;
```

**Response Structure** (`fuse_command_resp_t`):

```
typedef struct packed {
    data_t              data;       // Read data
    logic               status;     // Operation status
    logic               valid;      // Response valid
} fuse_command_resp_t;
```

**Command Types**:

- `FUSE_COMMAND_READ (2'b00)`: Read operation
- `FUSE_COMMAND_PROGRAM (2'b01)`: Program operation
- `FUSE_COMMAND_PROGRAM_READ_BACK (2'b10)`: Program with verification

### Memory Map and Register Reference

#### Memory Map

eFuse is not one universally placed address space. SEP, SMC, and the Key Manager pass-through expose different subsets and placements. The generated tables below compose those integration-specific views from their canonical SystemRDL roots.

*Table 1896. SEP eFuse Placement*

| Base Address | End Address | Size | Decoded Extent | Unit | Description |
|---|---|---|---|---|---|
| 0x10930000 | 0x109303FF | 1 KiB | 1 KiB | Shadow Registers | Cached eFuse shadow registers. |
| 0x10930400 | 0x109304FF | 256 B | 28 B | eFuse Control CSR | eFuse command and status registers. |
| 0x10930500 | 0x109305FF | 256 B | 116 B | MMR Registers | eFuse security-token registers. |

*Table 1897. SMC eFuse Placement*

| Address Range | Size | Decoded Extent | Unit | Description |
|---|---|---|---|---|
| BASE + 0x7000 – BASE + 0x7FFF | 4 KiB | 1 KiB | Shadow Registers | Address map for efuse shadow registers providing access to various chip configuration and ID fields |
| BASE + 0x8000 – BASE + 0x8FFF | 4 KiB | 28 B | eFuse Control CSR | eFuse command and status registers. |

*Table 1898. Key Manager eFuse Pass-Through*

| Base Address | End Address | Size | Decoded Extent | Unit | Description |
|---|---|---|---|---|---|
| 0x00011000 | 0x000113FF | 1 KiB | 1 KiB | Shadow Registers | SEP eFuse shadow-register pass-through. |

| Base Address | End Address | Size | Decoded Extent | Unit | Description |
|---|---|---|---|---|---|
| 0x00011400 | 0x000114FF | 256 B | 28 B | eFuse Control CSR | SEP eFuse command and status pass-through. |
| 0x00011500 | 0x000115FF | 256 B | 116 B | MMR Registers | SEP eFuse security-token pass-through. |

At the IP boundary, each region's base address and aperture size are integration parameters; the generated tables show the selected integration values. The MMR region exists only when the eFuse instance includes lifecycle state (`HAS_LC_STATE=1`).

### Detailed Register Map

### Shadow Registers

The shadow registers provide cached read access to eFuse contents, loaded automatically from the physical eFuse array during boot. The field layout is defined by the SystemRDL sources of the instantiating subsystem; the generated table below is authoritative for addresses, widths, and access rights of the instance documented in this section.

### Efuse address map of physical efuse macro

Address map for efuse shadow registers providing access to various chip configuration and ID fields

**NOTE**

**Register Arrays:**

- **I2C_I3C_ID[9]**: 9 registers, base 0x28, stride 0x8
- **SPARE[28]**: 28 registers, base 0x80, stride 0x20

*Table 1899. Efuse address map of physical efuse macro register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | LOCKS | RW | Lock controls for each fuse field. When write locked, the corresponding field is not programmable. When read locked, the shadow register is not readable. |
| 0x8 | JTAG_PUBLIC_IDENTITY | RW | JTAG-public exception window. In PROD and RMA_SiP, JTAG reads of this range are admitted while JTAG access to every other fuse field is blocked. Encoding is adopter-defined; the harness does not interpret or consume the value. The slot-0 read lock still hides the bits when set. |
| 0x28 - 0x68 | I2C_I3C_ID[9] | RW | I2C/I3C interface ID |
| 0x70 | SMC_CONFIG | RW | SMC boot and configuration controls sourced from fuse. Only the low word carries assigned bits, so firmware reads it as a single 32-bit access. |
| 0x78 | OCCP_TRANSPORT_TIMEOUT | RW | OCCP transport timeout in the low word; upper word reserved. |
| 0x80 - 0x3E0 | SPARE[28] | RW | Individually lockable 256-bit spare region, reserved for fields assigned after the RTL is frozen. Each element has its own write/read lock pair in LOCKS. |

### LOCKS

*Table 1900. smc_efuse_map.LOCKS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `spare27_read_lock` | RW1S | 0x0 | Spare 27 read lock (slot 31). When set, SPARE[27] is not readable. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 62 | spare27_write_lock | RW1S | 0x0 | Spare 27 write lock (slot 31). When set, SPARE[27] is not writable. |
| 61 | spare26_read_lock | RW1S | 0x0 | Spare 26 read lock (slot 30). When set, SPARE[26] is not readable. |
| 60 | spare26_write_lock | RW1S | 0x0 | Spare 26 write lock (slot 30). When set, SPARE[26] is not writable. |
| 59 | spare25_read_lock | RW1S | 0x0 | Spare 25 read lock (slot 29). When set, SPARE[25] is not readable. |
| 58 | spare25_write_lock | RW1S | 0x0 | Spare 25 write lock (slot 29). When set, SPARE[25] is not writable. |
| 57 | spare24_read_lock | RW1S | 0x0 | Spare 24 read lock (slot 28). When set, SPARE[24] is not readable. |
| 56 | spare24_write_lock | RW1S | 0x0 | Spare 24 write lock (slot 28). When set, SPARE[24] is not writable. |
| 55 | spare23_read_lock | RW1S | 0x0 | Spare 23 read lock (slot 27). When set, SPARE[23] is not readable. |
| 54 | spare23_write_lock | RW1S | 0x0 | Spare 23 write lock (slot 27). When set, SPARE[23] is not writable. |
| 53 | spare22_read_lock | RW1S | 0x0 | Spare 22 read lock (slot 26). When set, SPARE[22] is not readable. |
| 52 | spare22_write_lock | RW1S | 0x0 | Spare 22 write lock (slot 26). When set, SPARE[22] is not writable. |
| 51 | spare21_read_lock | RW1S | 0x0 | Spare 21 read lock (slot 25). When set, SPARE[21] is not readable. |
| 50 | spare21_write_lock | RW1S | 0x0 | Spare 21 write lock (slot 25). When set, SPARE[21] is not writable. |
| 49 | spare20_read_lock | RW1S | 0x0 | Spare 20 read lock (slot 24). When set, SPARE[20] is not readable. |
| 48 | spare20_write_lock | RW1S | 0x0 | Spare 20 write lock (slot 24). When set, SPARE[20] is not writable. |
| 47 | spare19_read_lock | RW1S | 0x0 | Spare 19 read lock (slot 23). When set, SPARE[19] is not readable. |
| 46 | spare19_write_lock | RW1S | 0x0 | Spare 19 write lock (slot 23). When set, SPARE[19] is not writable. |
| 45 | spare18_read_lock | RW1S | 0x0 | Spare 18 read lock (slot 22). When set, SPARE[18] is not readable. |
| 44 | spare18_write_lock | RW1S | 0x0 | Spare 18 write lock (slot 22). When set, SPARE[18] is not writable. |
| 43 | spare17_read_lock | RW1S | 0x0 | Spare 17 read lock (slot 21). When set, SPARE[17] is not readable. |
| 42 | spare17_write_lock | RW1S | 0x0 | Spare 17 write lock (slot 21). When set, SPARE[17] is not writable. |
| 41 | spare16_read_lock | RW1S | 0x0 | Spare 16 read lock (slot 20). When set, SPARE[16] is not readable. |
| 40 | spare16_write_lock | RW1S | 0x0 | Spare 16 write lock (slot 20). When set, SPARE[16] is not writable. |
| 39 | spare15_read_lock | RW1S | 0x0 | Spare 15 read lock (slot 19). When set, SPARE[15] is not readable. |
| 38 | spare15_write_lock | RW1S | 0x0 | Spare 15 write lock (slot 19). When set, SPARE[15] is not writable. |
| 37 | spare14_read_lock | RW1S | 0x0 | Spare 14 read lock (slot 18). When set, SPARE[14] is not readable. |
| 36 | spare14_write_lock | RW1S | 0x0 | Spare 14 write lock (slot 18). When set, SPARE[14] is not writable. |
| 35 | spare13_read_lock | RW1S | 0x0 | Spare 13 read lock (slot 17). When set, SPARE[13] is not readable. |
| 34 | spare13_write_lock | RW1S | 0x0 | Spare 13 write lock (slot 17). When set, SPARE[13] is not writable. |
| 33 | spare12_read_lock | RW1S | 0x0 | Spare 12 read lock (slot 16). When set, SPARE[12] is not readable. |
| 32 | spare12_write_lock | RW1S | 0x0 | Spare 12 write lock (slot 16). When set, SPARE[12] is not writable. |
| 31 | spare11_read_lock | RW1S | 0x0 | Spare 11 read lock (slot 15). When set, SPARE[11] is not readable. |
| 30 | spare11_write_lock | RW1S | 0x0 | Spare 11 write lock (slot 15). When set, SPARE[11] is not writable. |
| 29 | spare10_read_lock | RW1S | 0x0 | Spare 10 read lock (slot 14). When set, SPARE[10] is not readable. |
| 28 | spare10_write_lock | RW1S | 0x0 | Spare 10 write lock (slot 14). When set, SPARE[10] is not writable. |
| 27 | spare9_read_lock | RW1S | 0x0 | Spare 9 read lock (slot 13). When set, SPARE[9] is not readable. |
| 26 | spare9_write_lock | RW1S | 0x0 | Spare 9 write lock (slot 13). When set, SPARE[9] is not writable. |
| 25 | spare8_read_lock | RW1S | 0x0 | Spare 8 read lock (slot 12). When set, SPARE[8] is not readable. |
| 24 | spare8_write_lock | RW1S | 0x0 | Spare 8 write lock (slot 12). When set, SPARE[8] is not writable. |
| 23 | spare7_read_lock | RW1S | 0x0 | Spare 7 read lock (slot 11). When set, SPARE[7] is not readable. |
| 22 | spare7_write_lock | RW1S | 0x0 | Spare 7 write lock (slot 11). When set, SPARE[7] is not writable. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 21 | `spare6_read_lock` | RW1S | 0x0 | Spare 6 read lock (slot 10). When set, SPARE[6] is not readable. |
| 20 | `spare6_write_lock` | RW1S | 0x0 | Spare 6 write lock (slot 10). When set, SPARE[6] is not writable. |
| 19 | `spare5_read_lock` | RW1S | 0x0 | Spare 5 read lock (slot 9). When set, SPARE[5] is not readable. |
| 18 | `spare5_write_lock` | RW1S | 0x0 | Spare 5 write lock (slot 9). When set, SPARE[5] is not writable. |
| 17 | `spare4_read_lock` | RW1S | 0x0 | Spare 4 read lock (slot 8). When set, SPARE[4] is not readable. |
| 16 | `spare4_write_lock` | RW1S | 0x0 | Spare 4 write lock (slot 8). When set, SPARE[4] is not writable. |
| 15 | `spare3_read_lock` | RW1S | 0x0 | Spare 3 read lock (slot 7). When set, SPARE[3] is not readable. |
| 14 | `spare3_write_lock` | RW1S | 0x0 | Spare 3 write lock (slot 7). When set, SPARE[3] is not writable. |
| 13 | `spare2_read_lock` | RW1S | 0x0 | Spare 2 read lock (slot 6). When set, SPARE[2] is not readable. |
| 12 | `spare2_write_lock` | RW1S | 0x0 | Spare 2 write lock (slot 6). When set, SPARE[2] is not writable. |
| 11 | `spare1_read_lock` | RW1S | 0x0 | Spare 1 read lock (slot 5). When set, SPARE[1] is not readable. |
| 10 | `spare1_write_lock` | RW1S | 0x0 | Spare 1 write lock (slot 5). When set, SPARE[1] is not writable. |
| 9 | `spare0_read_lock` | RW1S | 0x0 | Spare 0 read lock (slot 4). When set, SPARE[0] is not readable. |
| 8 | `spare0_write_lock` | RW1S | 0x0 | Spare 0 write lock (slot 4). When set, SPARE[0] is not writable. |
| 7 | `occp_transport_timeout_read_lock` | RW1S | 0x0 | OCCP transport timeout read lock (slot 3). When set, OCCP_TRANSPORT_TIMEOUT is not readable. |
| 6 | `occp_transport_timeout_write_lock` | RW1S | 0x0 | OCCP transport timeout write lock (slot 3). When set, OCCP_TRANSPORT_TIMEOUT is not writable. |
| 5 | `smc_config_read_lock` | RW1S | 0x0 | SMC config read lock (slot 2). When set, SMC_CONFIG is not readable. |
| 4 | `smc_config_write_lock` | RW1S | 0x0 | SMC config write lock (slot 2). When set, SMC_CONFIG is not writable. |
| 3 | `i2c_i3c_id_read_lock` | RW1S | 0x0 | I2C/I3C ID read lock (slot 1). When set, no I2C_I3C_ID element is readable. |
| 2 | `i2c_i3c_id_write_lock` | RW1S | 0x0 | I2C/I3C ID write lock (slot 1). When set, no I2C_I3C_ID element is writable. |
| 1 | `jtag_public_identity_read_lock` | RW1S | 0x0 | JTAG public identity read lock (slot 0). When set, JTAG_PUBLIC_IDENTITY is not readable. |
| 0 | `jtag_public_identity_write_lock` | RW1S | 0x0 | JTAG public identity write lock (slot 0). When set, JTAG_PUBLIC_IDENTITY is not writable. |

### JTAG_PUBLIC_IDENTITY

*Table 1901. smc_efuse_map.JTAG_PUBLIC_IDENTITY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `value` | RW | 0x0 | Opaque payload. Encoding is adopter-defined; the harness does not interpret it. |

### I2C_I3C_ID[9]

*Table 1902. smc_efuse_map.I2C_I3C_ID[9] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `interface_id` | RW | 0x0 | I2C or I3C ID configuration: * Static ID mode: [6:0] contains ID, [63:7] reserved * Provisional ID mode: [63:0] full ID |

### SMC_CONFIG

*Table 1903. smc_efuse_map.SMC_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:16 | `config_rsvd_high` | RW | 0x0 | Reserved for future configuration bits. |
| 15 | `dft_ignore_error` | RW | 0x0 | When set, boot continues despite a reported memory repair or MBIST failure. |
| 14:8 | `config_rsvd_mid` | RW | 0x0 | Reserved for future configuration bits. |
| 7 | `sram_auto_zero_disable` | RW | 0x0 | When set, boot skips the SRAM auto-zero initialisation pass. |
| 6:1 | `config_rsvd_low` | RW | 0x0 | Reserved for future configuration bits. |
| 0 | `rom_flip_endianness` | RW | 0x0 | When set, the ROM interface swaps byte order on instruction fetch. |

#### OCCP_TRANSPORT_TIMEOUT

*Table 1904. smc_efuse_map.OCCP_TRANSPORT_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `timeout_rsvd` | RW | 0x0 | Reserved for extension of the timeout encoding. |
| 31:0 | `timeout` | RW | 0x0 | Transport timeout value consumed by the boot ROM OCCP driver. |

#### SPARE[28]

*Table 1905. smc_efuse_map.SPARE[28] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 255:0 | `rsvd` | RW | 0x0 | Spare fuse bits for future use. |

### MMR Registers

The MMR (Memory Mapped Register) space provides access to security tokens for authentication, including RMA tokens and security disable tokens.

#### eFuse Token Registers Address Map

Security-token input and match-status registers for RMA and secure-disable eFuse tokens.

**NOTE**

**Register Arrays:**

- **RMA_SIP_TOKEN_I[8]**: 8 registers, base 0x0, stride 0x4
- **RMA_CHIPLET_TOKEN_I[8]**: 8 registers, base 0x20, stride 0x4
- **SEC_DISABLE_TOKEN_I[8]**: 8 registers, base 0x40, stride 0x4

*Table 1906. eFuse Token Registers Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | RMA_SIP_TOKEN_I[8] | RW | Software-writeable RMA token word. The assembled token is hashed and compared against the fused value to authorize lifecycle (LC) state transitions. |
| 0x20 - 0x3C | RMA_CHIPLET_TOKEN_I[8] | RW | Software-writeable RMA token word. The assembled token is hashed and compared against the fused value to authorize lifecycle (LC) state transitions. |
| 0x40 - 0x5C | SEC_DISABLE_TOKEN_I[8] | RW | Software-writeable security-disable token word. The assembled token is hashed and compared against the post-silicon injected value to authorize security_disable. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x60 | TOKEN_EOP | W | Token end-of-packet control. Writing the per-token go bit signals that all token words have been written and triggers a pulse to begin token hashing. |
| 0x64 | RMA_SIP_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x68 | RMA_CHIPLET_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x6C | SEC_DISABLE_TOKEN_MATCH | R | Token match status. Reports the result of the token hash comparison (match, mismatch, or error). |
| 0x70 | TOKEN_MATCH_FAULT | R | Token comparator redundancy fault status. The token match logic is triple redundant and is not a majority vote: all three comparators must agree and each must drive a legal differential pair. A bit sets when the comparators for that token disagree, which may indicate a fault injection or FIB attack. The corresponding token_match_status then reads the error code 6'b111111 and an interrupt is raised to the SEP CPU. Bits are sticky until reset. |

### RMA_SIP_TOKEN_I[8]

*Table 1907. efuse_mmr.RMA_SIP_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of RMA_*_TOKEN, hashed, and compared with fused value to allow LC to change states. |

### RMA_CHIPLET_TOKEN_I[8]

*Table 1908. efuse_mmr.RMA_CHIPLET_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of RMA_*_TOKEN, hashed, and compared with fused value to allow LC to change states. |

### SEC_DISABLE_TOKEN_I[8]

*Table 1909. efuse_mmr.SEC_DISABLE_TOKEN_I[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `token` | RW | - | Portion of writeable version of SEC_DISABLE_TOKEN, hashed, and compared with the post silicon injected values to allow security_disable. |

### TOKEN_EOP

*Table 1910. efuse_mmr.TOKEN_EOP fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:17 | `Reserved` | - | - | Reserved |
| 16 | `secure_disable_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `rma_chiplet_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |
| 7:1 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `rma_sip_token_go` | W 1P | 0x0 | Write 1 to indicate that all token data has been written; this generates a pulse to begin token hashing. |

#### RMA_SIP_TOKEN_MATCH

*Table 1911. efuse_mmr.RMA_SIP_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

#### RMA_CHIPLET_TOKEN_MATCH

*Table 1912. efuse_mmr.RMA_CHIPLET_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

#### SEC_DISABLE_TOKEN_MATCH

*Table 1913. efuse_mmr.SEC_DISABLE_TOKEN_MATCH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:0 | `token_match_status` | R | 0x0 | Store the status of the token match. Match = 6'b010101, Mismatch = 6'b101010, Error = 6'b111111 |

#### TOKEN_MATCH_FAULT

*Table 1914. efuse_mmr.TOKEN_MATCH_FAULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:17 | `Reserved` | - | - | Reserved |
| 16 | `secure_disable_token_fault` | R | 0x0 | A redundancy fault was detected in the SEC_DISABLE_TOKEN comparators. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `rma_chiplet_token_fault` | R | 0x0 | A redundancy fault was detected in the RMA_CHIPLET_TOKEN comparators. |
| 7:1 | `Reserved` | - | - | Reserved |
| 0 | `rma_sip_token_fault` | R | 0x0 | A redundancy fault was detected in the RMA_SIP_TOKEN comparators. |

### eFuse Control Registers

The eFuse Control CSR provides command and status registers for read, write, and verification operations on the eFuse array. `program_enable` and `read_enable` are mutually exclusive. The eFuse command-path ownership rules in the Programmer's Guide give the order software follows.

### eFuse Interface Control Address Map

Program and read command, address, data, status, and timeout registers for the eFuse interface.

*Table 1915. eFuse Interface Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | EFUSE_INTERFACE_CTRL_STATUS | RW | eFuse interface status and error-clear register. Reports the sense-done flag and request/address error conditions, and provides write-1 strobes to clear those errors. |
| 0x4 | EFUSE_PROGRAM_CTRL | RW | eFuse program control and status. Configures the program address and data, triggers a program (optionally with read-back), and reports program progress and completion status. |
| 0x8 | EFUSE_READ_CTRL | RW | eFuse read control and status. Configures the read address, triggers a read, and reports read progress and completion status. |
| 0xC | EFUSE_PROGRAM_INTERFACE_READ_DATA | R | Read data captured through the program interface. Hardware debug only; valid only while the program interface state machine is in its capture-data state. |
| 0x10 | EFUSE_READ_INTERFACE_READ_DATA | R | Read data returned through the eFuse read interface. |
| 0x14 | EFUSE_READ_REQ_TIMEOUT | RW | Read interface timeout configuration. Sets the number of cycles the read interface waits for a SHIM response and enables the read timeout. |
| 0x18 | EFUSE_PROGRAM_REQ_TIMEOUT | RW | Program interface timeout configuration. Sets the number of cycles the program interface waits for a SHIM response and enables the program timeout. |

#### EFUSE_INTERFACE_CTRL_STATUS

*Table 1916. efuse_interface_ctrl.EFUSE_INTERFACE_CTRL_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10 | `efuse_read_addr_error_clear` | W 1P | 0x0 | Clears the read out of bounds error bit (efuse_read_addr_error) in efuse_ctrl_status |
| 9 | `efuse_program_addr_error_clear` | W 1P | 0x0 | Clears the program out of bounds error bit (efuse_program_addr_error) in efuse_ctrl_status |
| 8 | `efuse_req_error_clear` | W 1P | 0x0 | Clears the blocked request error bit in efuse_ctrl_status |
| 7 | `Reserved` | - | - | Reserved |
| 6 | `efuse_read_addr_error` | R | 0x0 | Indicates that an invalid eFuse read address was used (address exceeds the implemented eFuse size). The error must be cleared by writing 1 to efuse_read_addr_error_clear. |
| 5 | `efuse_program_addr_error` | R | 0x0 | Indicates that an invalid eFuse program address was used (address exceeds the implemented eFuse size). The error must be cleared by writing 1 to efuse_program_addr_error_clear. |
| 4 | `efuse_req_error` | R | 0x0 | Indicates that the eFuse was blocked because of either read or a program lock. The error must be cleared by writing 1 to efuse_req_error_clear to this register |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `efuse_sense_done` | R | - | Indicates if the eFuse state machine has completed |

#### EFUSE_PROGRAM_CTRL

*Table 1917. efuse_interface_ctrl.EFUSE_PROGRAM_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27 | `program_enable` | RW | 0x0 | When set, enables the program request |
| 26 | `program_status` | R | 0x0 | 0: No Error. 1: Error |
| 25 | `program_done` | R | 0x0 | When set, the program has completed, the status of which is in program_status |
| 24 | `program_busy` | R | 0x0 | When set, the program is in progress |
| 23:19 | `Reserved` | - | - | Reserved |
| 18 | `efuse_program_read_back` | RW | 0x0 | When set, trigger a read back after a program |
| 17 | `efuse_program_go` | RW 1P | 0x0 | When set, triggers a program to the eFuse. This will be ignored if the lock bit is set |
| 16 | `efuse_data` | RW | 0x0 | Data to program to eFuse. If this bit is 0, the program request will be ignored |
| 15:0 | `efuse_addr` | RW | 0x0 | Bit address to program |

### EFUSE_READ_CTRL

*Table 1918. efuse_interface_ctrl.EFUSE_READ_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `read_enable` | RW | 0x0 | When set, enables the read request |
| 27 | `Reserved` | - | - | Reserved |
| 26 | `read_status` | R | 0x0 | 0: No Error. 1: Error. Set when read_go is asserted while read is not enabled, the address is out of bounds, or the read request times out. |
| 25 | `read_done` | R | 0x0 | When set, the read has completed, the status of which is in read_status |
| 24 | `read_busy` | R | 0x0 | When set, the read is in progress |
| 23:17 | `Reserved` | - | - | Reserved |
| 16 | `efuse_read_go` | RW 1P | 0x0 | When set, triggers a read to efuse. |
| 15:0 | `efuse_addr` | RW | 0x0 | Bit address to read |

### EFUSE_PROGRAM_INTERFACE_READ_DATA

*Table 1919. efuse_interface_ctrl.EFUSE_PROGRAM_INTERFACE_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `dout` | R | 0x0 | Read Data from eFuse through program interface. This field is only valid when the efuse programming interface is utilized to burn the efuse IP and only when the state machine is during ST_CAPTUER_DATA state. Hardware debug only |

### EFUSE_READ_INTERFACE_READ_DATA

*Table 1920. efuse_interface_ctrl.EFUSE_READ_INTERFACE_READ_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `dout` | R | 0x0 | Read Data from eFuse through READ interface. |

#### EFUSE_READ_REQ_TIMEOUT

*Table 1921. efuse_interface_ctrl.EFUSE_READ_REQ_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `read_req_timout_enable` | RW | 0x0 | Enable timeout on read interface |
| 27:0 | `read_req_timeout_cycles` | RW | 0x800000 | Number of cycles the read interface will wait for a response from the shim. At 100MHz this can count ~= 2.5 seconds. |

#### EFUSE_PROGRAM_REQ_TIMEOUT

*Table 1922. efuse_interface_ctrl.EFUSE_PROGRAM_REQ_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28 | `program_req_timeout_enable` | RW | 0x0 | Enable timeout on program interface. |
| 27:0 | `program_req_timeout_cycles` | RW | 0x800000 | Number of cycles the program interface will wait for a response from the shim. At 100MHz this can count ~= 2.5 seconds. |

### SHIM Control Registers

The SHIM Control CSR provides foundry-specific configuration registers for timing adaptation and eFuse bank initialization. Its placement is adopter-defined within the adopter-owned SMC external aperture; it therefore does not appear at a fixed address in the generated OCAH map.

### eFuse SHIM Control Model Address Map

DV model of the eFuse SHIM control interface.

*Table 1923. eFuse SHIM Control Model Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | EFUSE_BANK_INIT_TIME | RW | eFuse bank initialization time. Number of cycles to wait for the OTP macro to initialize before sensing can begin. |

#### EFUSE_BANK_INIT_TIME

*Table 1924. efuse_shim_ctrl.EFUSE_BANK_INIT_TIME fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `init_time` | RW | 0x20 | Most OTP have a initialization time before sensing can be done |

## UART 16550

### Overview

The UART provides a 16550-style programming interface over 32-bit AXI4-Lite. It transmits and receives serial characters, buffers them in independently sized FIFOs, and reports line, modem and FIFO-integrity events. Extensions include larger FIFO depths, FIFO data/pointer parity protection, extended receive thresholds and a FIFO-error interrupt.

The TX and RX FIFOs default to 16 entries each. Software selects FIFO or single-character operation, frame format, baud divisor and interrupt enables. Modem signals provide status and software-controlled outputs; CTS does not stop

transmission and RTS does not automatically throttle reception. DMA-ready outputs indicate service conditions but do not themselves move data. The UART section of the Programmer's Guide gives the configuration, interrupt service and flow-control procedures.

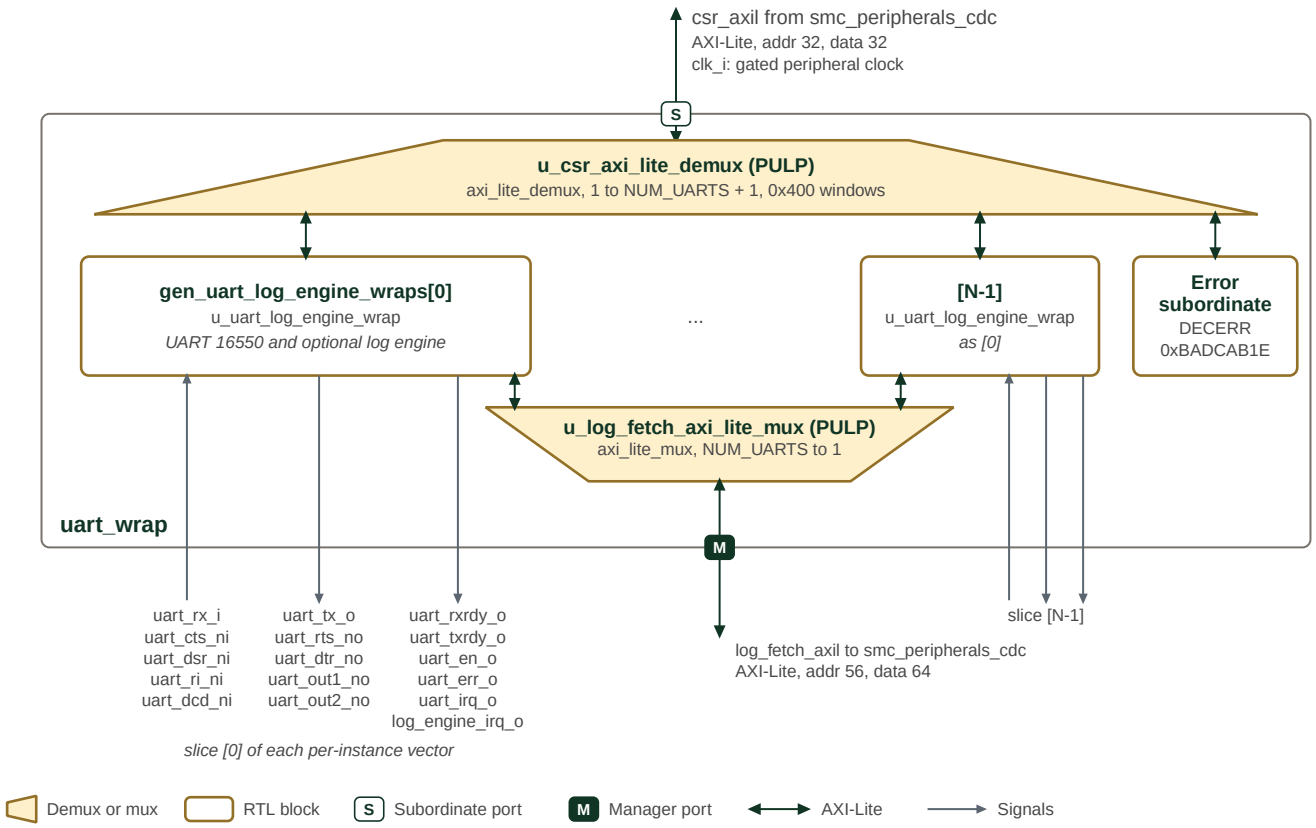


*Figure 66. SMC UART wrapper*

Figure 66 shows the SMC wrapper around the UART 16550. `uart_wrap` runs on one clock, `clk_i`, which in the SMC is the gated peripheral clock; both of its AXI4-Lite ports cross to the SMC clock in `smc_peripherals_cdc`. The register port has 32-bit address and data. The demux `u_csr_axi_lite_demux` and the mux `u_log_fetch_axi_lite_mux` are PULP blocks, marked by the project name in parentheses. The demux selects the instance whose window, 0x400 bytes apart, contains the address, and sends every other address to an error slave that answers DECERR with read data `0xBADCAB1E`.

Each instance, `gen_uart_log_engine_wraps[i].u_uart_log_engine_wrap`, holds one UART 16550 and, when bit `i` of `GEN_LOG_ENGINES` is set, a log engine; Figure 67 shows its contents. The mux `u_log_fetch_axi_lite_mux` merges the instances' log-fetch manager ports onto the single log-fetch port, with 56-bit address and 64-bit data. The SMC instantiates four UARTs, each with a log engine and 32-entry TX and RX FIFOs.

Every per-instance port is a vector with one bit per instance, and the figure labels the slice that instance 0 drives. In the SMC, `uart_dsr_ni`, `uart_ri_ni` and `uart_dcd_ni` are tied high, and `uart_dtr_no`, `uart_out1_no`, `uart_out2_no`, `uart_rxrdy_o` and `uart_txrdy_o` are unconnected. `uart_irq_o`, `uart_err_o` and `log_engine_irq_o` of each instance are ORed into one interrupt, peripheral interrupts 18 to 21.

### Architecture

`uart_16550` selects the register view; `uart_core` manages buffering, baud timing, modem state and interrupts; `uart_tx` and `uart_rx` serialize and sample characters. All core logic uses `clk_i`.

### Transmit and Receive Paths

With `LCR.DLAB=0`, a write to `THR` supplies a transmit byte and a read of `RBR` consumes a received byte. FIFO mode queues characters; non-FIFO mode uses one holding register per direction. Changing `FCR.FIFO_ENABLE` clears both FIFOs and the single-character holding state, so changing mode can discard data.

The transmitter runs whenever enabled and data is available, independently of CTS. `LSR.THRE` means the TX FIFO/holding register is empty; `LSR.TEMT` also requires the serializer to be idle. Use TEMT when the final character must have left the transmitter before reconfiguration.

The receiver synchronizes `rx_i` and takes a majority of three consecutive `clk_i` samples, which rejects single-cycle glitches. It then samples each bit once, at the bit midpoint timed by the 16-times-baud tick. Received parity, framing and break indications accompany characters; software reads line status before consuming their data. In FIFO mode, an overrun drops the newly received character; in non-FIFO mode an unread character can be overwritten in RBR. Frame format supports 5–8 data bits, selectable parity and stop-bit configuration. `LCR.SET_BREAK` forces the transmit line low.

### Baud Timing and Register Views

`LCR.DLAB` selects DLL/DLM at offsets `0x00` and `0x04` for divisor programming. With DLAB clear, these offsets select RBR/THR and IER. Other offsets retain their normal meaning. Serial operation is disabled when the combined divisor is zero; otherwise baud rate is `clk_i_frequency / (16 * (divisor + 1))`. The smallest usable divisor, 1, gives a maximum baud rate of `clk_i_frequency/32`.

### FIFO Protection and Line Errors

FIFO/holding-register data and FIFO pointers have parity protection. An internal parity failure drives `err_o` and can raise the separately enabled FIFO-error interrupt. This is different from parity in a received serial frame.

*Table 1925. UART serial errors*

| LSR field | Meaning |
|---|---|
| `OE` | Receive FIFO overrun: a character arrived while the receive FIFO was full. |
| `PE` | Parity error on the character exposed at the receive head. |
| `FE` | Invalid stop bit on that character. |
| `BI` | Break: the received frame remained low. |

In the current RTL, overwriting an unread non-FIFO RBR does not set OE. These error fields clear on an LSR read. Capture the status once for the character being serviced; repeated diagnostic reads can consume error evidence. The generated map specifies the status fields and their access side effects.

### Interrupt Priority and Service

`irq_o` reports the OR of the interrupt requests. `IIR.INTERRUPT_PENDING=0` means a request is pending; `IIR.INTERRUPT_ID` identifies the highest-priority source. The following table is ordered highest first; IDs are the three-bit field values, not the full IIR register value.

*Table 1926. UART interrupt priority and service*

| ID | Source | Service / control |
|---|---|---|
| 7 | FIFO integrity error | Investigate `err_o`; `IER.EFEI` masks this source. An IIR read does not repair or acknowledge an underlying parity failure. |
| 3 | Receiver line status | Capture LSR error fields, then service receive data; controlled by `IER.ELSI`. |

| ID | Source | Service / control |
|---|---|---|
| 6 | Receive timeout | Read RBR to clear the latched timeout. It is generated in FIFO mode when the receive FIFO holds data and its occupancy has not changed for roughly four character times. In the current RTL this request is not gated by IER. |
| 2 | Received data ready | Drain RBR below the selected FIFO trigger, or consume the non-FIFO character; controlled by `IER.ERBFI`. |
| 1 | Transmit holding register empty | Write THR, or read IIR while this is the selected source, to clear its latch. A subsequent transition back to empty re-arms it; controlled by `IER.ETBEI`. |
| 0 | Modem status | Read MSR once to capture levels and clear delta bits; controlled by `IER.EDSSI`. |

`MCR.OUT2` does not gate `irq_o`; its modem and loopback uses are described in the UART modem status section of the Programmer's Guide. The ITR extension can force interrupt sources for testing. Clear test settings after use. Do not rely on clearing IER alone to silence the unmasked timeout source; use the subsystem interrupt controller when masking all delivery is required.

### Interface

#### Configuration Parameters

The UART 16550 module provides configurable parameters to customize FIFO sizes according to system requirements.

The parameters and ports of `uart_16550` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Modem Control Interface

The modem control interface provides full RS-232 modem signaling compatibility. All modem signals are active low at the pins, matching the RS-232 convention.

The modem inputs are **status only**. Each one is synchronized and then reported in the `MSR` register in the opposite polarity (a bit reads `1` while its pin is asserted low), and a change on any of them raises the Modem Status Interrupt. None of them gates the datapath: this component implements no automatic flow control, so software must act on the interrupt to throttle traffic. The software flow control procedure is in the Programmer's Guide.

#### DMA Interface

The DMA interface provides signals for Direct Memory Access controllers to manage data transfers automatically. The `rxrdy_o` and `txrdy_o` signals indicate when DMA operations can proceed, with behavior configurable through FIFO control modes.

### Interface Timing Considerations

#### Clock Domain

All interface signals operate in a single clock domain referenced to `clk_i`. The following timing considerations apply:

- **AXI4-Lite Protocol**: All AXI4-Lite signals must meet setup and hold time requirements relative to `clk_i`
- **Asynchronous Inputs**: `rx_i` and modem control inputs (`cts_ni`, `dsr_ni`, `ri_ni`, `dcd_ni`) are asynchronous and are synchronized internally
- **Reset Requirements**: `rst_ni` should be asserted asynchronously and deasserted synchronously relative to `clk_i`

### Serial Interface Timing

- **Baud Rate**: Serial data rate is determined by the divisor latch registers (DLL/DLM) and uses 16x oversampling
- **Maximum Baud Rate**: Limited to `system_clock_frequency / 32` - a divisor of `0` disables the transmitter and receiver, so the smallest usable divisor is `1`
- **Data Frame**: Configurable start bit (always 1), 5-8 data bits, optional parity bit, 1-2 stop bits

### Protocol Support

The interface supports standard UART communication protocols:

- **RS-232**: Full modem control and serial data signaling
- **RS-485**: Can be used with external transceivers (using RTS for direction control)
- **Industrial Protocols**: Compatible with Modbus RTU and other industrial serial protocols
- **Debug Interfaces**: Suitable for system debug and console applications

The comprehensive interface design ensures compatibility with a wide range of serial communication requirements while maintaining the flexibility needed for diverse embedded system applications.

## Memory Map and Register Reference

### Memory Map

The core decodes a 64-byte register space with 32-bit registers on four-byte boundaries. Offsets `0x00` and `0x04` change meaning with `LCR.DLAB`; offset `0x08` reads IIR and writes FCR. See Programming for safe divisor and FIFO configuration. The generated views below retain all field definitions, including ECR extended trigger selection and ITR interrupt tests.

### Detailed Register Map

The first two register views apply when `LCR.DLAB=0`; read-only and write-only aliases at the same offset are shown separately. Registers outside offsets `0x00` and `0x04`, including FCR at `0x08`, are accessible for either `LCR.DLAB` value. The divisor-latch view applies when `LCR.DLAB=1`.

### UART 16550 Main Address Map

Contains the mode-independent registers and the read-only registers accessible only when `DLAB = 0`.

*Table 1927. UART 16550 Main Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | RBR | R | Receiver Buffer Register |
| 0x4 | IER | RW | Interrupt Enable Register |
| 0x8 | IIR | R | Interrupt Identification Register |
| 0xC | LCR | RW | Line Control Register |
| 0x10 | MCR | RW | Modem Control Register |
| 0x14 | LSR | R | Line Status Register |
| 0x18 | MSR | R | Modem Status Register |
| 0x1C | SCR | RW | Scratch Register |
| 0x20 | ECR | RW | Extended Control Register |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x24 | ITR | RW | Interrupt Test Register |

### RBR

*Table 1928. uart_16550_main.RBR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R RMOD | 0x0 | Received Data. Contains a received character. If FIFOs are enabled, this field points to the bottom of the RX FIFO. Otherwise, this field points to a single- byte Receiver Buffer Register. |

### IER

*Table 1929. uart_16550_main.IER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `EFEI` | RW | 0x0 | Enable FIFO Error Interrupt. |
| 3 | `EDSSI` | RW | 0x0 | Enable (Delta Status of) Modem Status Interrupt. |
| 2 | `ELSI` | RW | 0x0 | Enable (Receiver) Line Status Interrupt. |
| 1 | `ETBEI` | RW | 0x0 | Enable Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. |
| 0 | `ERBFI` | RW | 0x0 | Enable Receiver Buffer Full (Received Data Ready) Interrupt. |

### IIR

*Table 1930. uart_16550_main.IIR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:6 | `FIFOS_ENABLED` | R | 0x0 | FIFOs Enabled: * `0x0` - FIFOs are disabled * `0x3` - FIFOs are enabled |
| 5:4 | `Reserved` | - | - | Reserved |
| 3:1 | `INTERRUPT_ID` | R | 0x0 | Interrupt ID: * `0x7` - FIFO Error Interrupt (priority 0) * `0x3` - Receiver Line Status Interrupt (priority 1) * `0x6` - Reception Timeout Interrupt (priority 2) * `0x2` - Received Data Ready Interrupt (priority 3) * `0x1` - Transmitter Holding Register Empty Interrupt (priority 4) * `0x0` - Modem Status Interrupt (priority 5) |
| 0 | `INTERRUPT_PENDING` | R | 0x1 | Interrupt Pending. Active-low. |

### LCR

*Table 1931. uart_16550_main.LCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `DLAB` | RW | 0x0 | Divisor Latch Access Bit. If set, allows access to the `DLL` and `DLM` registers when accessing addresses `0x0` and `0x4`, respectively. If unset, allows access to the `THR`, `RBR`, and `IIR` registers. |
| 6 | `SET_BREAK` | RW | 0x0 | Set Break. If set, forces the `tx_o` output to `0` to cause a break condition on the receiving UART. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 5 | STICK_PARITY | RW | 0x0 | Stick Parity. If set, forces the transmit and received parity bits to be 0 if even parity is selected and 1 if odd parity is selected. |
| 4 | EPS | RW | 0x0 | Even Parity Select: * 1 - even parity * 0 - odd parity |
| 3 | PEN | RW | 0x0 | Parity Enable. |
| 2 | STB | RW | 0x0 | Stop Bits: * 0 - 1 stop bit. * 1 - 2 stop bits (or 1.5 stop bits if word length is set to 5 bits) |
| 1:0 | WLS | RW | 0x0 | Word Length Select: * 0x0 - 5 bits per character * 0x1 - 6 bits per character * 0x2 - 7 bits per character * 0x3 - 8 bits per character |

### MCR

*Table 1932. uart_16550_main.MCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5 | LINE_LOOPBACK | RW | 0x0 | Line Loopback. If set, the rx_i input is internally connected to the tx_o output. |
| 4 | LOOP | RW | 0x0 | System Loopback. If set, the transmitter is internally connected to the receiver. The tx_o output is set to 1. |
| 3 | OUT2 | RW | 0x0 | User Output 2. Writing to this bit drives the out2_no in the opposite polarity. |
| 2 | OUT1 | RW | 0x0 | User Output 1. Writing to this bit drives the out1_no in the opposite polarity. |
| 1 | RTS | RW | 0x0 | Request to Send. Writing to this bit drives the rts_no output in the opposite polarity. |
| 0 | DTR | RW | 0x0 | Data Terminal Ready. Writing to this bit drives the dtr_no output in the opposite polarity. |

### LSR

*Table 1933. uart_16550_main.LSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7 | ERROR_IN_RCVR_FIFO | R | 0x0 | Error in Receiver FIFO. When set, indicates that the receiver FIFO or (RBR in non-FIFO mode) has the parity, framing, or break error. In other words, at least one of the PE, FE, and BI bits is set. |
| 6 | TEMT | R | 0x1 | Transmitter Empty. When set, indicates that the transmitter shift register and the transmitter FIFO (or THR in non-FIFO mode) are empty. |
| 5 | THRE | R | 0x1 | Transmitter Holding Register Empty. When set, indicates that the TX FIFO (or THR in non-FIFO mode) is empty. Clearing the Transmitter Holding Register Empty Interrupt does NOT clear this bit. |
| 4 | BI | R RC | 0x0 | Break Interrupt. When set, indicates that the rx_i input is 0 for an entire frame's time (start + data + parity + stop bits). In FIFO mode, this bit set when the character reaches the top of the receiver FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 3 | FE | R RC | 0x0 | Framing Error. When set, indicates that the received character is missing a stop bit. In FIFO mode, this bit is set when the character reaches the top of the FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 2 | PE | R RC | 0x0 | Parity Error. When set, indicates that the character at the top of the receiver FIFO (or RBR in non-FIFO mode) has a parity error. This bit triggers the Receiver Line Status Interrupt. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | OE | R RC | 0x0 | Overrun Error. When set, indicates that the RX FIFO (or Receiver Buffer Register in Non-FIFO Mode) is full when new character is received. In FIFO Mode, the new character is dropped. In non-FIFO mode, the existing character in the RBR is overwritten. This bit triggers the Receiver Line Status Interrupt. |
| 0 | DR | R | 0x0 | Data Ready. When set, indicates that the RX FIFO (or RBR in non-FIFO mode) contains data. |

### MSR

*Table 1934. uart_16550_main.MSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7 | DCD | R | 0x0 | Data Carrier Detect. This bit reflects the dcd_ni input in the opposite polarity, so it reads 1 while a carrier is detected. |
| 6 | RI | R | 0x0 | Ring Indicator. This bit reflects the ri_ni input in the opposite polarity, so it reads 1 while the remote modem signals an incoming call. |
| 5 | DSR | R | 0x0 | Data Set Ready. This bit reflects the dsr_ni input in the opposite polarity, so it reads 1 while the remote modem is connected and ready. |
| 4 | CTS | R | 0x0 | Clear to Send. This bit reflects the cts_ni input in the opposite polarity, so it reads 1 while the remote device is ready to receive data. |
| 3 | DDCD | R RC | 0x0 | Delta Data Carrier Detect. When set, indicates that the DCD bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 2 | TERI | R RC | 0x0 | Trailing Edge Ring Indicator. When set, indicates that the RI bit has changed from a 1 to a 0. This bit triggers the Modem Status Interrupt. |
| 1 | DDSR | R RC | 0x0 | Delta Data Set Ready. When set, indicates that the DSR bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 0 | DCTS | R RC | 0x0 | Delta Clear to Send. When set, indicates that the CTS bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |

### SCR

*Table 1935. uart_16550_main.SCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | SCR | RW | 0x0 | Scratch. Holds user data. |

### ECR

*Table 1936. uart_16550_main.ECR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1:0 | RCVR_TRIGGER_MS2B | RW | 0x0 | Receiver FIFO Trigger Level Most Significant 2 Bits. To configure the least significant 2 bits, use the FCR.RCVR_TRIGGER register field. The configurations for the trigger levels are: * 0x0 - 1 character * 0x1 - 4 characters * 0x2 - 8 characters * 0x3 - 14 characters * 0x4 - 32 characters * 0x5 - 64 characters * 0x6 - 128 characters * 0x7 - 256 characters * 0x8 - 512 characters * 0x9 - 1024 characters * 0xA - 2048 characters * 0xB - 4096 characters |

### ITR

*Table 1937. uart_16550_main.ITR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5 | TRTI | RW | 0x0 | Test Reception Timeout Interrupt. Writing a 1 forces the interrupt and writing 0 releases it. |
| 4 | TFEI | RW | 0x0 | Test FIFO Error Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 3 | TDSSI | RW | 0x0 | Test (Delta Status of) Modem Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 2 | TLSI | RW | 0x0 | Test (Receiver) Line Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 1 | TTBEI | RW | 0x0 | Test Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 0 | TRBFI | RW | 0x0 | Test Receiver Buffer Full (Received Data Ready) Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |

### UART 16550 Main Write-Only Address Map

Contains the write-only registers. THR is accessible only when LCR.DLAB = 0; FCR is writable for either LCR.DLAB value.

*Table 1938. UART 16550 Main Write-Only Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | THR | W | Transmitter Holding Register |
| 0x8 | FCR | W | FIFO Control Register |

### THR

*Table 1939. uart_16550_main_wo.THR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | DATA | W WMOD | 0x0 | Transmit Data. Holds a character to be transmitted. If FIFOs are enabled, this field points to the top of the TX FIFO. Otherwise, this field points to a single-byte Transmitter Holding Register. |

### FCR

*Table 1940. uart_16550_main_wo.FCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:6 | RCVR_TRIGGER | W | 0x0 | Receiver FIFO Trigger Level (Least Significant 2 Bits). To configure the most significant 2 bits, use the ECR.RCVR_TRIGGER_MS2B register field. The configurations for the trigger levels are: * 0x0 - 1 character * 0x1 - 4 characters * 0x2 - 8 characters * 0x3 - 14 characters * 0x4 - 32 characters * 0x5 - 64 characters * 0x6 - 128 characters * 0x7 - 256 characters * 0x8 - 512 characters * 0x9 - 1024 characters * 0xA - 2048 characters * 0xB - 4096 characters |
| 5:4 | Reserved | - | - | Reserved |
| 3 | DMA_MODE_SELECT | W | 0x0 | DMA mode select: * 0 - DMA Mode 0 * 1 - DMA Mode 1 |
| 2 | XMIT_FIFO_RESET | W 1P | 0x0 | Transmitter FIFO Reset. Writing 1 clears the TX FIFO. |
| 1 | RCVR_FIFO_RESET | W 1P | 0x0 | Receiver FIFO Reset. Writing 1 clears the RX FIFO. |
| 0 | FIFO_ENABLE | W | 0x0 | FIFO Enable. When set, enables the TX and RX FIFOs (FIFO Mode). When unset, disables the FIFOs (Non-FIFO Mode). |

### UART 16550 Divisor Latch Address Map

Contains the registers accessible only when LCR.DLAB = 1.

*Table 1941. UART 16550 Divisor Latch Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | DLL | RW | Divisor Latch Least Significant Byte Register |
| 0x4 | DLM | RW | Divisor Latch Most Significant Byte Register |

### DLL

*Table 1942. uart_16550_dl.DLL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | DLL | RW | 0x0 | Divisor Latch Least Significant Byte. Baud rate is calculated as baud_rate = system_clock_frequency / (16 * (divisor + 1)). When the divisor is set to 0, the transmitter and receiver logic are disabled. |

### DLM

*Table 1943. uart_16550_dl.DLM fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | DLM | RW | 0x0 | Divisor Latch Most Significant Byte. Baud rate is calculated as baud_rate = system_clock_frequency / (16 * (divisor + 1)). When the divisor is set to 0, the transmitter and receiver logic are disabled. |

## Log Engine

### Overview

The Log Engine sends memory-resident messages to a UART without a CPU copying each byte. Software supplies one memory region, divided into 16 slots, and submits each message by writing its slot's length. The engine selects pending slots in round-robin order and completes one message before moving to the next.

A 64-bit AXI4-Lite read path fetches message data into a FIFO. A separate 32-bit AXI4-Lite write path sends one character at a time to a fixed UART address, paced by `uart_tx_ready_i`. The register interface is also 32-bit AXI4-Lite. Fetch addresses are 56 bits; UART addresses are 32 bits. The Log Engine section of the Programmer's Guide gives the configuration, submission and recovery procedures.

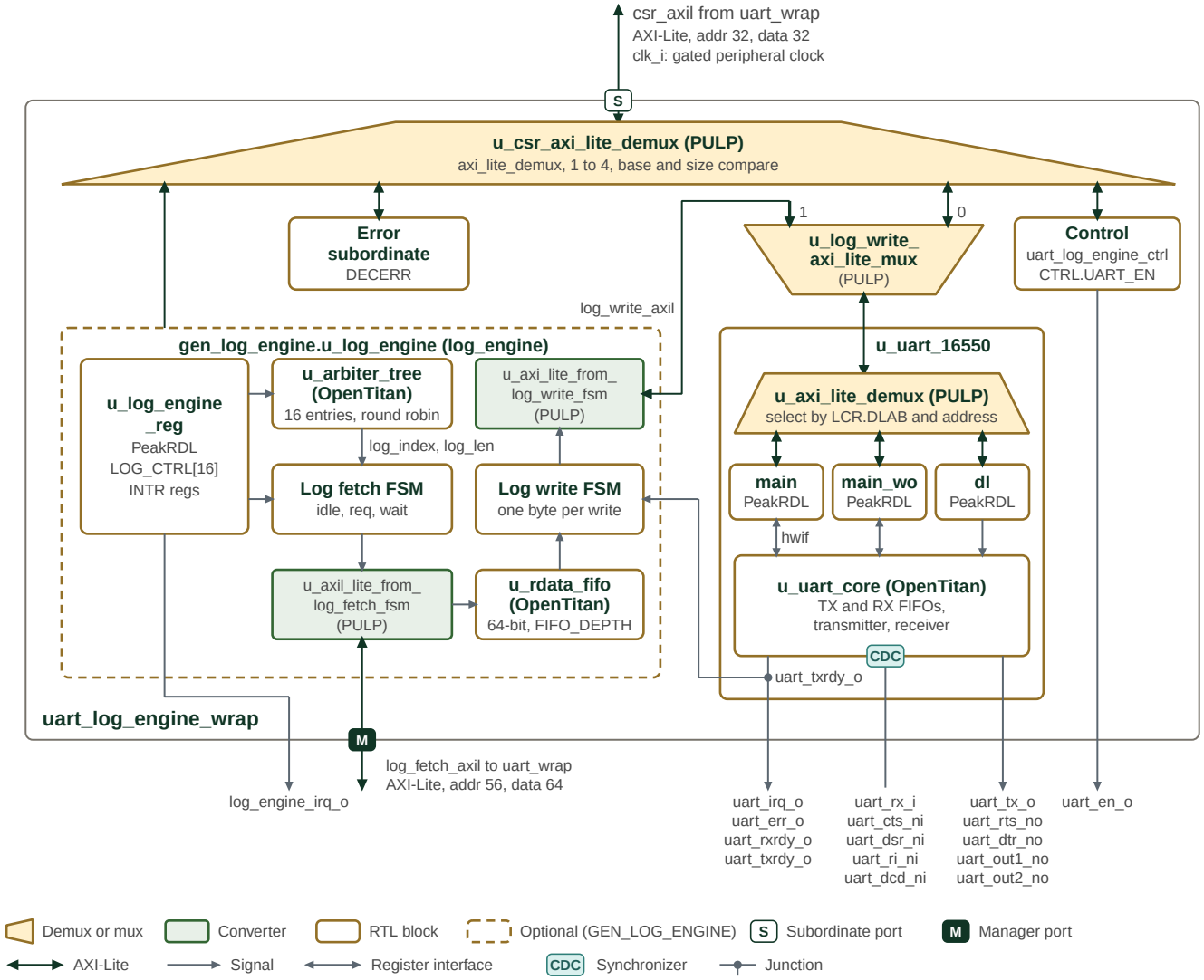


*Figure 67. SMC log-engine wrapper and data paths*

Figure 67 shows the SMC wrapper and its memory-to-UART data paths. Parentheses after an instance name give its module or, for a block taken from another project, that project: the demuxes, the mux and the `axi_lite_from_mem` converters come from PULP, and `u_uart_core`, `u_arbiter_tree` and `u_rdata_fifo` from OpenTitan. `uart_log_engine_wrap` runs on one clock, `clk_i`. Its register port, with 32-bit address and data, comes from `uart_wrap`, and the demux `u_csr_axi_lite_demux` decodes it by base and size onto four maps: the UART, the log engine, the control block and an error slave that answers DECERR. The control block `u_uart_log_engine_ctrl_reg`, generated by PeakRDL from `uart_log_engine_ctrl.rdl`, drives `uart_en_o` from `CTRL.UART_EN`.

The UART register path and the log engine's write path share the UART's register port through the mux `u_log_write_axi_lite_mux`: port 0 carries register accesses and port 1 the log engine's writes. Inside `u_uart_16550`, the demux `u_axi_lite_demux` selects one of three PeakRDL register blocks. While `LCR.DLAB` is set, `DLL` and `DLM` go to the divisor-latch block (`dl`); writes to `THR` and `FCR` go to the write-only block (`main_wo`); everything else goes to the main block. `u_uart_core` holds the TX and RX FIFOs, the transmitter and the receiver, and passes `uart_rx_i` and the modem inputs through two-flop synchronizers, marked by the synchronizer badge on its edge.

The dashed outline marks the log engine, which exists only when `GEN_LOG_ENGINE` is set; without it the log-fetch port is driven to zero and `log_engine_irq_o` is low. `u_log_engine_reg`, generated from `log_engine.rdl`, holds the 16 `LOG_CTRL` registers and the interrupt registers, and drives `log_engine_irq_o`. `u_arbiter_tree` picks a log entry whose `LOG_LEN` is nonzero and gives its index and length to the log fetch FSM. The fetch FSM reads 64-bit words through the `axi_lite_from_mem` converter `u_axil_lite_from_log_fetch_fsm` on the log-fetch port, which has 56-bit address and 64-bit data, into `u_rdata_fifo`, whose depth is `FIFO_DEPTH` (4 in the SMC). The log write FSM takes one byte at a time from the FIFO and writes it, through a second converter, `u_axi_lite_from_log_write_fsm`, to `LOG_WRITE_ADDR` over a 32-bit AXI4-Lite port. It waits for `uart_txrdy_o` before each write.

### Operation

#### Memory Slots and Lengths

`LOG_REGION_ADDR` specifies the start of a shared region. `LOG_REGION_SIZE` specifies its total size in bytes. Use an 8-byte-aligned base and a nonzero size aligned to 128 bytes, up to 512 KiB. The region is divided equally among 16 slots; it does not provide independently sized buffers.

For supported settings, slot `i` starts at:

```
LOG_REGION_ADDR + (LOG_REGION_SIZE / 16) * i
```

A nonzero `LOG_CTRL[i].LOG_LEN` makes that slot pending. The length is in bytes and remains the submitted value until hardware clears it on completion; it is not a bytes-remaining counter. Keep the buffer and its configuration unchanged until completion.

For oversized regions, hardware clamps the effective size to 512 KiB. For a size not aligned to 128 bytes, it divides the effective size by 16 and rounds each slot's transferable capacity down to a multiple of eight. Requested lengths above that capacity are clamped. A zero-capacity slot completes without a transfer. Software should validate size and length instead of relying on truncation.

#### Arbitration and Data Transfer

The arbiter selects among nonzero lengths and advances when the selected message completes. Scheduling is per message, not per byte. A stalled UART or bus response holds up that message and all later entries; round-robin selection does not impose a completion-time bound.

The fetch FSM issues single-beat 64-bit reads and waits for each response. A `FIFO_DEPTH`-entry FIFO buffers the returned words. The final read still fetches a full eight-byte word when the message length is not a multiple of eight, so the whole fetched word must be readable within the slot.

The write FSM sends bytes in increasing address order, least-significant byte first within each fetched word. It waits for FIFO data and `uart_tx_ready_i`, writes the character in data bits `[7:0]` to `LOG_WRITE_ADDR`, then waits for the response. The current implementation sets all four write strobes; the UART target must accept that access.

Completion clears the selected length and advances arbitration. It means the write responses have returned, not that the UART has finished shifting the final character onto its serial output.

#### Errors and Interrupts

`INTR_STATUS.LOG_FETCH_ERR` (bit 0) and `LOG_WRITE_ERR` (bit 4) latch bus response errors whether or not their interrupts are enabled. `INTR_ENABLE` masks the corresponding contributions to `irq_o`. Write 1 to a status bit to clear it; `INTR_TEST` can

force either condition.

The FSMs advance on error responses as well as successful responses. There is no automatic retry or per-slot success flag, and a cleared length alone does not prove successful delivery. Software must inspect error status and choose recovery appropriate to its logging requirements.

### Reset and Disable

All logic uses `clk_i`. Active-low `rst_ni` asynchronously resets the registers, state machines, counters and FIFO. Reset release timing is the integrator's responsibility; the module does not synchronize it internally.

Clearing `CTRL.EN` returns the fetch/write FSMs and counters to idle and clears the FIFO. It does not clear configuration or pending lengths, and it does not reset the AXI adapters. Disabling an active transfer is not a guaranteed drain or clean cancellation. Configure while idle; following an aborted transfer, reconcile outstanding bus traffic before reusing buffers or restarting the engine.

### Interface

The parameters and ports of `log_engine` are listed in the HTML edition of this manual, under RTL Modules Reference.

*Table 1944. Log-engine configuration*

| Setting | Value | Source |
|---|---|---|
| `FIFO_DEPTH` | 4 by default | Module parameter: buffered 64-bit words. |
| `NumLogEntries` | 16 | Package constant and generated register inventory. |
| Maximum region | 512 KiB | Package constant `MaxLogRegionSize`. |
| Fetch address/data | 56/64 bits | Package types. |
| Write address/data | 32/32 bits | Package types. |
| CSR data | 32 bits | Register interface; address width comes from the generated map. |

A shared `clk_i` and active-low asynchronous `rst_ni` drive all interfaces and internal logic. The CSR AXI4-Lite slave (`csr_axil_req_i` / `csr_axil_resp_o`) carries register traffic; two AXI4-Lite initiators carry memory reads (`log_fetch_axil_req_o` / `log_fetch_axil_resp_i`) and UART writes (`log_write_axil_req_o` / `log_write_axil_resp_i`). Both initiator adapters use `PROT = 3'b010` (non-secure, unprivileged data) and allow one outstanding request each. They issue no bursts.

`uart_tx_ready_i` permits the write FSM to request the next character transfer. The UART integration supplies readiness in the engine's clock domain and must account for writes already accepted by the bus. See Arbitration and Data Transfer for the write data and strobe behavior.

`irq_o` is the active-high OR of enabled, latched fetch and write error status.

### Memory Map and Register Reference

### Memory Map

Offsets below are relative to the Log Engine register bank. All registers reset to zero. The generated map gives field access and side effects.

*Table 1945. Log-engine register summary*

| Register | Offset | Purpose |
|---|---|---|
| `CTRL` | `0x00` | Engine enable. |
| `LOG_REGION_SIZE` | `0x04` | Total region size in bytes. |
| `LOG_REGION_ADDR` | `0x08/0x0C` | Low/high halves of the 56-bit fetch base. |
| `LOG_WRITE_ADDR` | `0x10` | UART write address. |
| `INTR_STATUS` | `0x14` | Latched fetch/write errors; write 1 to clear. |
| `INTR_ENABLE` | `0x18` | Error interrupt enables. |
| `INTR_TEST` | `0x1C` | Force error status for interrupt testing. |
| `LOG_CTRL[i]` | `0x40 + 4*i` | Submitted byte length for slot 0–15; cleared on completion. |

### Detailed Register Map

### Log Engine Address Map

Configuration and status registers for the UART log engine, including the log region and write pointers.

**NOTE**

**Register Arrays:**

- **LOG_CTRL[16]**: 16 registers, base 0x40, stride 0x4

*Table 1946. Log Engine Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CTRL | RW | Control Register |
| 0x4 | LOG_REGION_SIZE | RW | Log Region Size Register |
| 0x8 | LOG_REGION_ADDR | RW | Log Region Address Register |
| 0x10 | LOG_WRITE_ADDR | RW | Log Write Address Register |
| 0x14 | INTR_STATUS | RW | Interrupt Status Register |
| 0x18 | INTR_ENABLE | RW | Interrupt Enable Register |
| 0x1C | INTR_TEST | W | Interrupt Test Register |
| 0x40 - 0x7C | LOG_CTRL[16] | RW | Log Control Register |

### CTRL

*Table 1947. log_engine.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `EN` | RW | 0x0 | Log Engine Enable. When set, enables the Log Engine. When cleared, the engine stops fetching and writing, discards buffered log data and resets transfer progress. Configuration and pending LOG_LEN values are retained, so a pending entry restarts from the start of its slot when the engine is enabled again. Clearing does not wait for outstanding bus responses. |

### LOG_REGION_SIZE

*Table 1948. log_engine.LOG_REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `LOG_REGION_SIZE` | RW | 0x0 | Log region size in bytes. Supported values are at most 512 KiB and aligned to 128 bytes (16 slots times the 8-byte fetch beat). If software programs a larger value, RTL clamps the effective region to 512 KiB. For a value that is not 128-byte aligned, RTL divides the effective region among the 16 slots and rounds each slot's transferable capacity down to complete 8-byte fetch beats. A requested LOG_LEN above that capacity is clamped to the capacity. |

### LOG_REGION_ADDR

*Table 1949. log_engine.LOG_REGION_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `RESERVED` | R | 0x0 | Reserved. Reads as zero; writes are ignored. |
| 55:32 | `LOG_REGION_ADDR_HI` | RW | 0x0 | Log Region Start Address High. Contains fabric address bits [55:32]. |
| 31:0 | `LOG_REGION_ADDR_LO` | RW | 0x0 | Log Region Start Address Low. |

### LOG_WRITE_ADDR

*Table 1950. log_engine.LOG_WRITE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `LOG_WRITE_ADDR` | RW | 0x0 | Log Write Address. This is the address where the log data is written to. |

### INTR_STATUS

*Table 1951. log_engine.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `LOG_WRITE_ERR` | RW1C | 0x0 | Log Write Error Interrupt. Asserted when the bus for writing log data returns an error. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `LOG_FETCH_ERR` | RW1C | 0x0 | Log Fetch Error Interrupt. Asserted when the bus for fetching log data returns an error. |

### INTR_ENABLE

*Table 1952. log_engine.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `LOG_WRITE_ERR` | RW | 0x0 | Log Write Error Interrupt Enable. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `LOG_FETCH_ERR` | RW | 0x0 | Log Fetch Error Interrupt Enable. |

### INTR_TEST

*Table 1953. log_engine.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `LOG_WRITE_ERR` | W 1P | 0x0 | Log Write Error Interrupt Test. Writing `1` forces the interrupt. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `LOG_FETCH_ERR` | W 1P | 0x0 | Log Fetch Error Interrupt Test. Writing `1` forces the interrupt. |

### LOG_CTRL[16]

*Table 1954. log_engine.LOG_CTRL[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `LOG_LEN` | RW | 0x0 | Log Length. Writing a nonzero value representing the log length in bytes to this field starts the log transfer process. The field holds the requested length until the transfer completes, when hardware clears it to zero; it is not a remaining-byte counter. There are 16 copies of this register, each representing a log entry. The log entries go through round robin arbitration. |

## I2C Controller

The I2C controller is based on the OpenTitan I2C design. Controller and target behavior, clock stretching, arbitration, timing setup, and programming sequences are specified in the OpenTitan theory of operation and the OpenTitan programmer's guide.

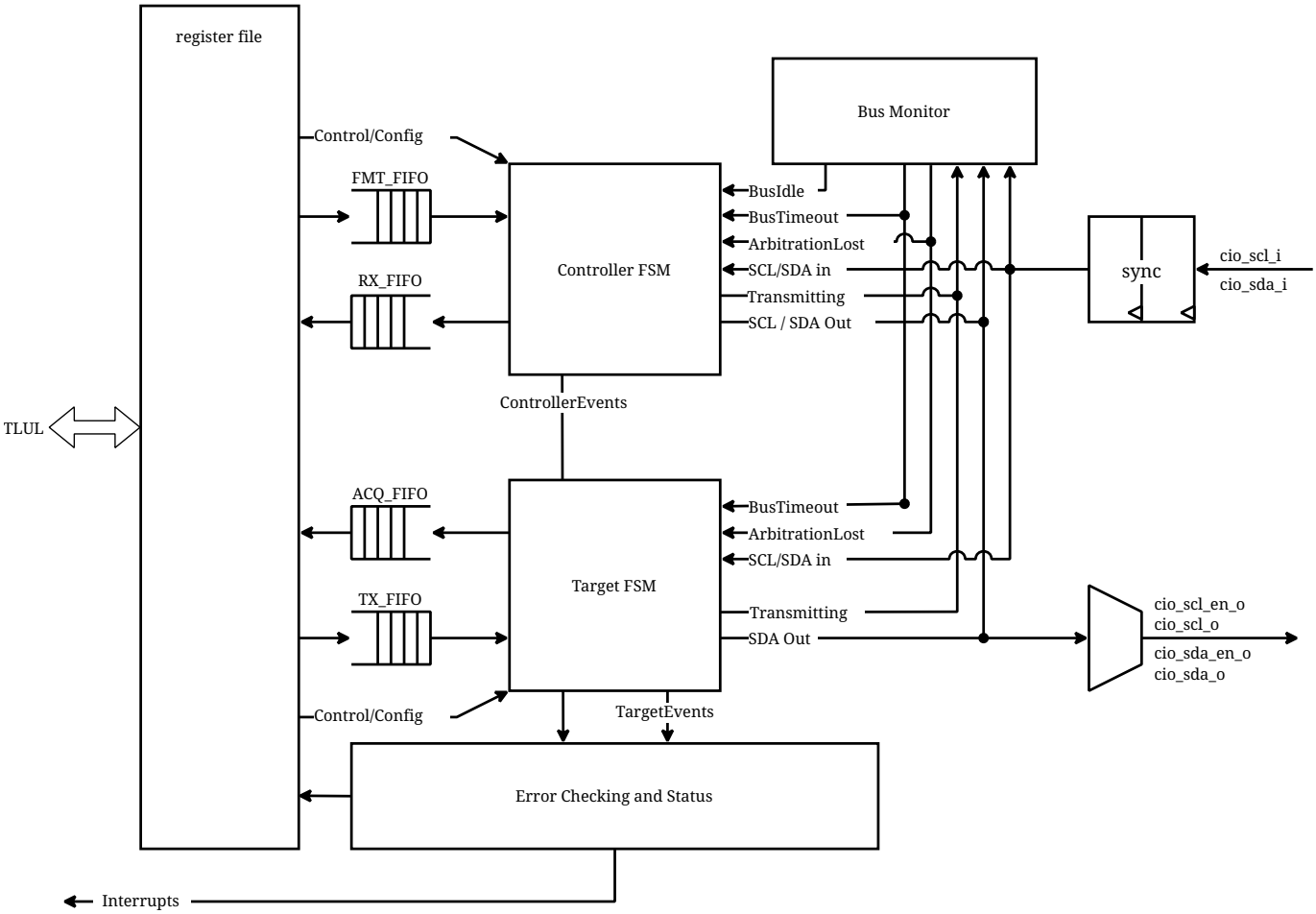


*Figure 68. OpenTitan I2C core reference architecture*

Figure 68 shows the upstream core.

### OCAH Modifications

- AXI4-Lite register interface in place of the OpenTitan TileLink host port.
- Wrapper register blocks `i2c_ctrl` and `i2c_wrap` around the OpenTitan register bank.

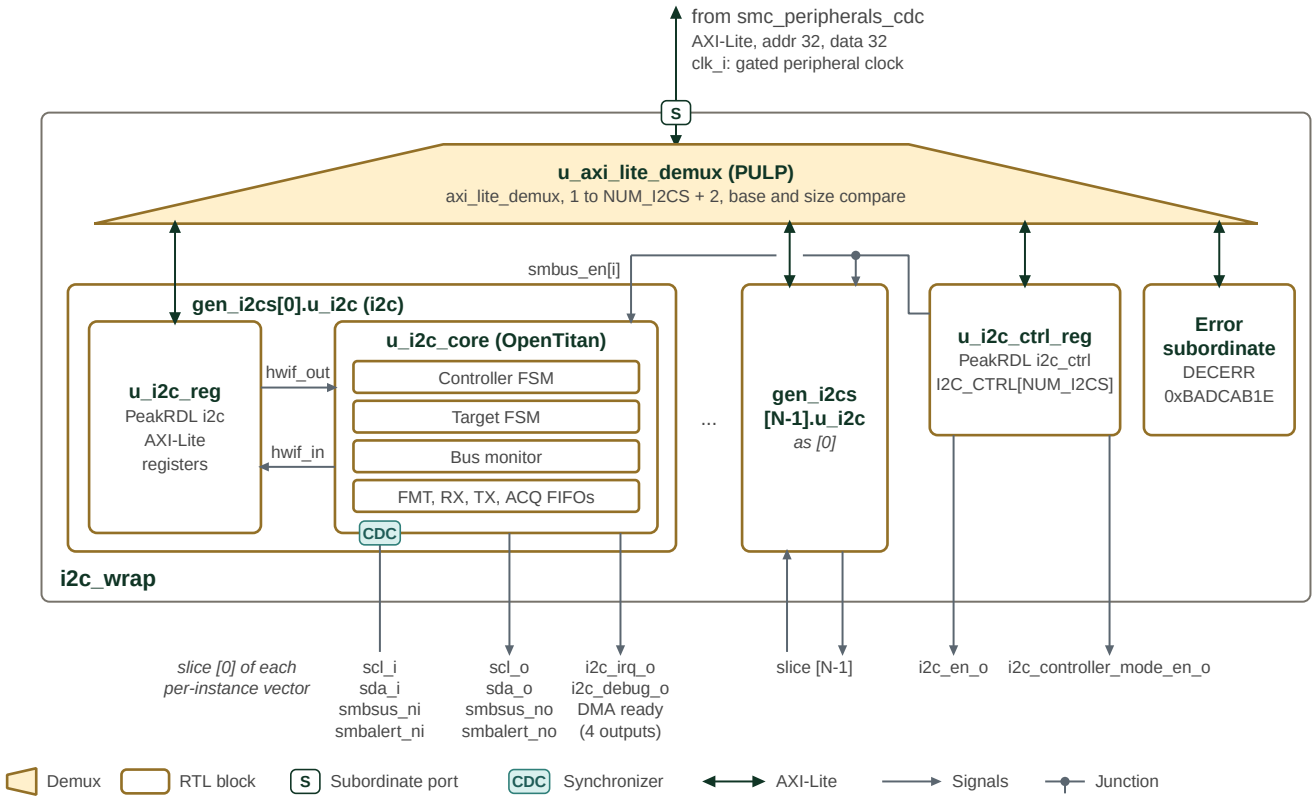


*Figure 69. OCAH I2C wrapper and generated register interface*

Figure 69 shows the OCAH wrapper around that core. `i2c_wrap` has one AXI4-Lite slave port, with 32-bit address and data, and runs on a single clock, `clk_i`; in the SMC that is the gated peripheral clock, and the bus reaches it through `smc_peripherals_cdc`. Parentheses after an instance name give its module or, for a block taken from another project, that project. The demux `u_axi_lite_demux`, a PULP `axi_lite_demux`, compares the address against the base and size of each register map and selects one of `NUM_I2CS` + 2 ports: one per instance, the shared control map, and an error slave that answers DECERR with read data `0xBADCAB1E`. The SMC sets `NUM_I2CS` to 3.

Each instance, `gen_i2cs[i].u_i2c`, is the OpenTitan core `u_i2c_core` with its register bank replaced by `u_i2c_reg`, an AXI4-Lite register block that PeakRDL generates from `i2c.rdl`. The register block drives the core through `hwif_out` and takes status back through `hwif_in`. The core keeps the OpenTitan controller and target FSMs, bus monitor and FMT, RX, TX and ACQ FIFOs, sized by the `*_FIFO_DEPTH` parameters, and synchronizes the SCL, SDA, SMBus SUS and SMBus ALERT inputs to `clk_i` with two-flop synchronizers; the synchronizer badge on the edge of `u_i2c_core` marks them. The wrapper adds no synchronizers of its own.

The shared block `u_i2c_ctrl_reg`, generated from `i2c_ctrl.rdl`, holds one `I2C_CTRL` register per instance. Its `SMBUS_EN` field drives that instance's `smbus_en` input, which masks `smbalert_ni` while low. Its `I2C_EN` and `I2C_CONTROLLER_MODE_EN` fields leave the wrapper on `i2c_en_o` and `i2c_controller_mode_en_o` without reaching the instance. The SMC padring uses `I2C_EN` to select the I2C function on the instance's SCL, SDA, SMBus alert and SMBus suspend pads, and `I2C_CONTROLLER_MODE_EN` to set the direction of the SMBus alert and suspend pads. Neither field enables the core: its controller or target role comes from `CTRL.ENABLEHOST` and `CTRL.ENABLETARGET`, and `I2C_CONTROLLER_MODE_EN` should agree with that role.

Every per-instance port of `i2c_wrap` is a vector with one bit, or for `i2c_debug_o` one nibble, per instance. The figure labels the slice that instance 0 drives. The DMA ready outputs are `controller_tx_ready_o`, `controller_rx_ready_o`, `target_tx_ready_o` and `target_rx_ready_o`; the SMC leaves them unconnected. It routes `i2c_irq_o` to peripheral interrupts 23 to 25.

### Memory Map and Register Reference

#### Memory Map

The I2C Interface Component provides a comprehensive register interface through an 8-bit address space (0x00-0xFF) accessible via AXI4-Lite protocol. The register map is organized into logical groups covering interrupt management, control configuration, data access, timing parameters, and mode-specific functionality. All registers are 32-bit aligned and support both controller and target operating modes.

#### Address Alignment and Access Width

**Address Alignment:** - All registers are aligned to 32-bit boundaries (addresses are multiples of 4) - 8-bit effective address space: only bits [7:0] of the AXI4-Lite address are used - Upper address bits are ignored

**Access Width Requirements:** - All registers must be accessed using 32-bit operations - Byte and halfword accesses are not supported - AXI4-Lite write strobe signals should be set to 4'b1111 for all writes

**Endianness:** - All multi-byte fields use little-endian byte ordering - Bit numbering follows standard convention (bit 0 is LSB)

#### Detailed Register Map

#### I2C Address Map

Interrupt status, enable, and test registers for a single I2C controller and target.

*Table 1955. I2C Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | INTR_STATE | RW | Interrupt Status Register |
| 0x4 | INTR_ENABLE | RW | Interrupt Enable Register |
| 0x8 | INTR_TEST | RW | Interrupt Test Register |
| 0xC | SMBUS_CTRL | RW | SMBus Control Register |
| 0x10 | CTRL | RW | Control Register |
| 0x14 | STATUS | R | Status Register |
| 0x18 | RDATA | R | Controller RX FIFO Access Register (Controller Mode only) |
| 0x1C | FDATA | W | Controller TX FIFO Access Register (Controller Mode only) |
| 0x20 | FIFO_CTRL | W | FIFO Control Register |
| 0x24 | HOST_FIFO_CONFIG | RW | Controller FIFO Configuration Register (Controller Mode only) |
| 0x28 | TARGET_FIFO_CONFIG | RW | Target FIFOs Configuration Register (Target Mode only) |
| 0x2C | HOST_FIFO_STATUS | R | Controller FIFOs Status Register (Controller Mode only) |
| 0x30 | TARGET_FIFO_STATUS | R | Target FIFOs Status Register (Target Mode only) |
| 0x34 | OVRD | RW | Override Control Register |
| 0x38 | VAL | R | Bus Oversampled Values Register |
| 0x3C | TIMING0 | RW | SCL LOW and HIGH Periods Register |
| 0x40 | TIMING1 | RW | Bus Rise and Fall Times Register |
| 0x44 | TIMING2 | RW | START Condition Setup and Hold Times Register |
| 0x48 | TIMING3 | RW | Data Setup and Hold Times Register |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x4C | TIMING4 | RW | STOP Condition Setup Time and Bus Free Time Register |
| 0x50 | TIMEOUT_CTRL | RW | Timeout Control Register |
| 0x54 | TARGET_ID | RW | Target ID Register (Target Mode only) |
| 0x58 | ACQDATA | R | Target RX FIFO Access Register (Target Mode only) |
| 0x5C | TXDATA | W | Target TX FIFO Access Register (Target Mode only). |
| 0x60 | HOST_TIMEOUT_CTRL | RW | Controller Timeout Control Register |
| 0x64 | TARGET_TIMEOUT_CTRL | RW | Target Timeout Control Register (Target Mode only) |
| 0x68 | TARGET_NACK_COUNT | RW | Target NACK Count Register (Target Mode only) |
| 0x6C | TARGET_ACK_CTRL | RW | Target ACK Control Register (Target Mode only) |
| 0x70 | ACQ_FIFO_NEXT_DATA | R | Target RX FIFO Next Byte Register (Target Mode only) |
| 0x74 | HOST_NACK_HANDLER_TIMEOUT | RW | Controller NACK Timeout Register (Controller Mode only) |
| 0x78 | CONTROLLER_EVENTS | RW | Controller Events Register (Controller Mode only) |
| 0x7C | TARGET_EVENTS | RW | Target Events Register (Target Mode) |
| 0x80 | SMBUS_STATUS | R | SMBus Status Register |

### INTR_STATE

*Table 1956. i2c.INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19 | `TARGET_RX_FIFO_ERROR` | RW1C | 0x0 | Target Mode interrupt: asserted when the Target RX FIFO has a parity error. Write `1` to clear. |
| 18 | `TARGET_TX_FIFO_ERROR` | RW1C | 0x0 | Target Mode interrupt: asserted when the Target TX FIFO has a parity error. Write `1` to clear. |
| 17 | `CONTROLLER_RX_FIFO_ERROR` | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller RX FIFO has a parity error. Write `1` to clear. |
| 16 | `CONTROLLER_TX_FIFO_ERROR` | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller TX FIFO has a parity error. Write `1` to clear. |
| 15 | `SMBALERT` | RW1C | 0x0 | Controller Mode interrupt: asserted when `smbalert_ni` is asserted. Write `1` to clear. |
| 14 | `HOST_TIMEOUT` | RW1C | 0x0 | Target Mode interrupt: asserted when the controller stops generating the clock longer than `HOST_TIMEOUT_CTRL.VAL`. Write `1` to clear. |
| 13 | `UNEXP_STOP` | RW1C | 0x0 | Target Mode interrupt: asserted when the controller sends this target a STOP before this target NACKs. Write `1` to clear. |
| 12 | `ACQ_STRETCH` | R | 0x0 | Target Mode interrupt: remains asserted while the target is stretching the clock because 1) the Target RX FIFO is full or 2) the `TARGET_ACK_CTRL.NBYTES` count has reached `0` (only if `CTRL.ACK_CTRL_EN = 1`). |
| 11 | `TX_THRESHOLD` | R | 0x0 | Target Mode interrupt: remains asserted while the Target TX FIFO level is less than `TARGET_FIFO_CONFIG.TX_THRESH`. |
| 10 | `TX_STRETCH` | R | 0x0 | Target Mode interrupt: remains asserted while this target is stretching the clock or has halted. The flags in the `TARGET_EVENTS` register explain the reason(s) for clock stretching/halting. Clearing the `TARGET_EVENTS` flags clears this interrupt. |
| 9 | `CMD_COMPLETE` | RW1C | 0x0 | Controller/Target Mode interrupt: asserted when this/the controller finishes generating a STOP or repeated START. Write `1` to clear. |
| 8 | `SDA_UNSTABLE` | RW1C | 0x0 | Controller Mode interrupt: asserted when the target fails to keep SDA stable during a transmission. Write `1` to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7 | STRETCH_TIMEOUT | RW1C | 0x0 | Controller Mode interrupt: asserted when the target stretches the clock longer than TIMEOUT_CTRL.VAL (valid only when TIMEOUT_CTRL.MODE = 0). Write 1 to clear. |
| 6 | SDA_INTERFERENCE | RW1C | 0x0 | Controller Mode interrupt: asserted when SDA is unexpectedly pulled LOW by another controller. Write 1 to clear. |
| 5 | SCL_INTERFERENCE | RW1C | 0x0 | Controller Mode interrupt: asserted when SCL is unexpectedly pulled LOW by another controller. Write 1 to clear. |
| 4 | CONTROLLER_HALT | R | 0x0 | Controller Mode interrupt: remains asserted while this controller halts. The flags in the CONTROLLER_EVENTS register explain the reason(s) for the halting. Clearing the CONTROLLER_EVENTS flags clears this interrupt. |
| 3 | RX_OVERFLOW | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller RX FIFO overflows. Write 1 to clear. |
| 2 | ACQ_THRESHOLD | R | 0x0 | Target Mode interrupt: remains asserted while the Target RX FIFO level is greater than TARGET_FIFO_CONFIG.ACQ_THRESH. |
| 1 | RX_THRESHOLD | R | 0x0 | Controller Mode interrupt: remains asserted while the Controller RX FIFO level is greater than HOST_FIFO_CONFIG.RX_THRESH. |
| 0 | FMT_THRESHOLD | R | 0x0 | Controller Mode interrupt: remains asserted while the Controller TX FIFO level is less than HOST_FIFO_CONFIG.FMT_THRESH. |

### INTR_ENABLE

*Table 1957. i2c.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | TARGET_RX_FIFO_ERROR | RW | 0x0 | Enables the TARGET_RX_FIFO_ERROR interrupt. |
| 18 | TARGET_TX_FIFO_ERROR | RW | 0x0 | Enables the TARGET_TX_FIFO_ERROR interrupt. |
| 17 | CONTROLLER_RX_FIFO_ERROR | RW | 0x0 | Enables the CONTROLLER_RX_FIFO_ERROR interrupt. |
| 16 | CONTROLLER_TX_FIFO_ERROR | RW | 0x0 | Enables the CONTROLLER_TX_FIFO_ERROR interrupt. |
| 15 | SMBALERT | RW | 0x0 | Enables the SMBALERT interrupt. |
| 14 | HOST_TIMEOUT | RW | 0x0 | Enables the HOST_TIMEOUT interrupt. |
| 13 | UNEXP_STOP | RW | 0x0 | Enables the UNEXP_STOP interrupt. |
| 12 | ACQ_STRETCH | RW | 0x0 | Enables the ACQ_STRETCH interrupt. |
| 11 | TX_THRESHOLD | RW | 0x0 | Enables the TX_THRESHOLD interrupt. |
| 10 | TX_STRETCH | RW | 0x0 | Enables the TX_STRETCH interrupt. |
| 9 | CMD_COMPLETE | RW | 0x0 | Enables the CMD_COMPLETE interrupt. |
| 8 | SDA_UNSTABLE | RW | 0x0 | Enables the SDA_UNSTABLE interrupt. |
| 7 | STRETCH_TIMEOUT | RW | 0x0 | Enables the STRETCH_TIMEOUT interrupt. |
| 6 | SDA_INTERFERENCE | RW | 0x0 | Enables the SDA_INTERFERENCE interrupt. |
| 5 | SCL_INTERFERENCE | RW | 0x0 | Enables the SCL_INTERFERENCE interrupt. |
| 4 | CONTROLLER_HALT | RW | 0x0 | Enables the CONTROLLER_HALT interrupt. |
| 3 | RX_OVERFLOW | RW | 0x0 | Enables the RX_OVERFLOW interrupt. |
| 2 | ACQ_THRESHOLD | RW | 0x0 | Enables the ACQ_THRESHOLD interrupt. |
| 1 | RX_THRESHOLD | RW | 0x0 | Enables the RX_THRESHOLD interrupt. |
| 0 | FMT_THRESHOLD | RW | 0x0 | Enables the FMT_THRESHOLD interrupt. |

### INTR_TEST

*Table 1958. i2c.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19 | `TARGET_RX_FIFO_ERROR` | W 1P | 0x0 | Writing `1` forces the `TARGET_RX_FIFO_ERROR` interrupt. |
| 18 | `TARGET_TX_FIFO_ERROR` | W 1P | 0x0 | Writing `1` forces the `TARGET_TX_FIFO_ERROR` interrupt. |
| 17 | `CONTROLLER_RX_FIFO_ERROR` | W 1P | 0x0 | Writing `1` forces the `CONTROLLER_RX_FIFO_ERROR` interrupt. |
| 16 | `CONTROLLER_TX_FIFO_ERROR` | W 1P | 0x0 | Writing `1` forces the `CONTROLLER_TX_FIFO_ERROR` interrupt. |
| 15 | `SMBALERT` | W 1P | 0x0 | Writing `1` forces the `SMBALERT` interrupt. |
| 14 | `HOST_TIMEOUT` | W 1P | 0x0 | Writing `1` forces the `HOST_TIMEOUT` interrupt. |
| 13 | `UNEXP_STOP` | W 1P | 0x0 | Writing `1` forces the `UNEXP_STOP` interrupt. |
| 12 | `ACQ_STRETCH` | RW | 0x0 | Writing `1` forces the `ACQ_STRETCH` interrupt. Writing `0` releases it. |
| 11 | `TX_THRESHOLD` | RW | 0x0 | Writing `1` forces the `TX_THRESHOLD` interrupt. Writing `0` releases it. |
| 10 | `TX_STRETCH` | RW | 0x0 | Writing `1` forces the `TX_STRETCH` interrupt. Writing a `0` releases it. |
| 9 | `CMD_COMPLETE` | W 1P | 0x0 | Writing `1` forces the `CMD_COMPLETE` interrupt. |
| 8 | `SDA_UNSTABLE` | W 1P | 0x0 | Writing `1` forces the `SDA_UNSTABLE` interrupt. |
| 7 | `STRETCH_TIMEOUT` | W 1P | 0x0 | Writing `1` forces the `STRETCH_TIMEOUT` interrupt. |
| 6 | `SDA_INTERFERENCE` | W 1P | 0x0 | Writing `1` forces the `SDA_INTERFERENCE` interrupt. |
| 5 | `SCL_INTERFERENCE` | W 1P | 0x0 | Writing `1` forces the `SCL_INTERFERENCE` interrupt. |
| 4 | `CONTROLLER_HALT` | RW | 0x0 | Writing `1` forces the `CONTROLLER_HALT` interrupt. Writing `0` releases it. |
| 3 | `RX_OVERFLOW` | W 1P | 0x0 | Writing `1` forces the `RX_OVERFLOW` interrupt. |
| 2 | `ACQ_THRESHOLD` | RW | 0x0 | Writing `1` forces the `ACQ_THRESHOLD` interrupt. Writing `0` releases it. |
| 1 | `RX_THRESHOLD` | RW | 0x0 | Writing `1` forces the `RX_THRESHOLD` interrupt. Writing `0` releases it. |
| 0 | `FMT_THRESHOLD` | RW | 0x0 | Writing `1` forces the `FMT_THRESHOLD` interrupt. Writing `0` releases it. |

### SMBUS_CTRL

*Table 1959. i2c.SMBUS_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `SMBALERT` | RW | 0x0 | Target Mode control: asserts the `smbalert_no` output. This bit clears itself when the controller addresses this target. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `SMBSUS` | RW | 0x0 | Controller Mode control: asserts the `smbsus_no` output. |

### CTRL

*Table 1960. i2c.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7 | `ACQ_START_STOP_EN` | RW | 0x0 | ACQ FIFO Start/Stop Enable (Target Mode only): * `0` - Start/Stop symbols are NOT written into the ACQ FIFO. The firmware should rely on `TARGET_EVENTS.START_DETECT` and `TARGET_EVENTS.STOP_DETECT` flags to detect Start/Stop events. * `1` - Start/Stop symbols ARE written into the ACQ FIFO (legacy behavior). When a START (or repeated START) is detected, an `ACQ_START` or `ACQ_RESTART` entry is written. When a STOP is detected, an `ACQ_STOP` or `ACQ_NACK_STOP` entry is written. |
| 6 | `TX_STRETCH_CTRL_EN` | RW | 0x0 | Target mode control: enables the Software TX Stretch Control Mechanism. If this bit is: * `0` - Automatic TX Stretch: When this target receives a READ address byte, it only stretches the clock if the TX FIFO is empty; otherwise, it pops the FIFO and transmits the data byte. The target never sets the `TARGET_EVENTS` register flags in this mode. * `1` - Software TX Stretch Mode: When this target receives a READ address, it always stretches the clock and sets the `TARGET_EVENTS.TX_PENDING` flag. The software can: 1. confirm the release and transmission of the TX FIFO data by writing a 1 to clear the `TARGET_EVENTS.TX_PENDING` flag; or 2. reset the TX FIFO by writing 1 to `FIFO_CTRL.TXRST` and load in new data via the `TXDATA` register—useful whenthe READ address is targeting a different function of the target. In this mode, the target always sets the `TARGET_EVENTS` register flags. |
| 5 | `MULTI_CONTROLLER_MONITOR_EN` | RW | 0x0 | Global control: enables the Bus Monitor. Set this bit to `1` only in a multi-controller environment.<br><br>If a `0→`1` transition happens while `ENABLEHOST` and `ENABLETARGET` are both `0`, the Bus Monitor will enable and begin in the 'Bus Busy' state. To transition to a 'Bus Free' state, `HOST_TIMEOUT_CTRL` must be nonzero so the Bus Monitor may count out idle cycles to confirm the freedom to transmit. In addition, the Bus Monitor will track whether the bus is free based on the enabled timeouts and detected STOP symbols. For Multi-Controller Mode, ensure `MULTI_CONTROLLER_MONITOR_EN` becomes `1` no later than `ENABLEHOST` or `ENABLETARGET`. This bit can be set at the same time as either or both of the other two, though.<br><br>Note that if `MULTI_CONTROLLER_MONITOR_EN` is set after `ENABLEHOST` or `ENABLETARGET`, the Bus Monitor will begin in the 'Bus Free' state instead. This would violate the proper protocol for a controller to join a multi-controller environment. However, if this controller is known to be the first to join, this ordering will enable skipping the idle wait.<br><br>When `0`, the bus monitor will report that the bus is always free, so the Controller FSM is never blocked from transmitting. |
| 4 | `ACK_CTRL_EN` | RW | 0x0 | Target Mode control: enables the Software ACK Control Mechanism. If this bit is: * `0` - This target ACKs a data byte whenever the ACQ FIFO has space. * `1` - This target ACKs the first `TARGET_ACK_CTRL.NBYTES` bytes. If another byte arrives, this target stretches the clock and awaits software intervention (and asserts `STATUS.ACK_CTRL_STRETCH`). The software can 1. accept the new byte(s) by reloading the `TARGET_ACK_CTRL.NBYTES` counter; or 2. reject the new byte(s) by writing `1` to `TARGET_ACK_CTRL.NACK` (useful for SMBus). |
| 3 | `NACK_ADDR_AFTER_TIMEOUT` | RW | 0x0 | Target Mode control: NACK address after timeout. If this bit is: * `0` - This target ACKs the address byte even if a Stretch Timeout occurs (useful for SMBus). * `1` - This target NACKs the address byte when a Stretch Timeout occurs. |
| 2 | `LLPBK` | RW | 0x0 | Global control: enables line loop-back. In Controller Mode, the internal logic sees the `smbalert_ni` input as deasserted. In Target Mode, this target sends all received SDA data back out, and the internal logic sees received data as all 1's. |
| 1 | `ENABLETARGET` | RW | 0x0 | Global control: configures this device's operating mode. * `ENABLEHOST = 1` and `ENABLETARGET = 0` — Controller Mode * `ENABLEHOST = 0` and `ENABLETARGET = 1` — Target Mode * `ENABLEHOST = 1` and `ENABLETARGET = 1` — Hybrid Mode * `ENABLEHOST = 0` and `ENABLETARGET = 0` — Monitor Mode |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `ENABLEHOST` | RW | 0x0 | Global control: configures this device's operating mode. * `ENABLEHOST = 1` and `ENABLETARGET = 0` — Controller Mode * `ENABLEHOST = 0` and `ENABLETARGET = 1` — Target Mode * `ENABLEHOST = 1` and `ENABLETARGET = 1` — Hybrid Mode * `ENABLEHOST = 0` and `ENABLETARGET = 0` — Monitor Mode |

### STATUS

*Table 1961. i2c.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |
| 10 | `ACK_CTRL_STRETCH` | R | 0x0 | Target Mode status: indicates that this target is stretching the clock due to the Software ACK Control Mechanism. See `CTRL.ACK_CTRL_EN` for details. |
| 9 | `ACQEMPTY` | R | 0x1 | Target Mode status: Target RX FIFO empty. |
| 8 | `TXEMPTY` | R | 0x1 | Target Mode status: Target TX FIFO empty. |
| 7 | `ACQFULL` | R | 0x0 | Target Mode status: Target RX FIFO full. |
| 6 | `TXFULL` | R | 0x0 | Target Mode status: Target TX FIFO full. |
| 5 | `RXEMPTY` | R | 0x1 | Controller Mode status: Controller RX FIFO empty. |
| 4 | `TARGETIDLE` | R | 0x1 | Target Mode status: Target FSM idle. |
| 3 | `HOSTIDLE` | R | 0x1 | Controller Mode status: Controller FSM idle. |
| 2 | `FMTEMPTY` | R | 0x1 | Controller Mode status: Controller TX FIFO Empty. |
| 1 | `RXFULL` | R | 0x0 | Controller Mode status: Controller RX FIFO Full. |
| 0 | `FMTFULL` | R | 0x0 | Controller Mode status: Controller TX FIFO Full. |

### RDATA

*Table 1962. i2c.RDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R RMOD | 0x0 | Reading this register pops the Controller RX FIFO. |

### FDATA

*Table 1963. i2c.FDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12 | `NAKOK` | W | 0x0 | NACK OK. When set, this controller will not care if this byte is NACKed. It will not halt, set the `CONTROLLER_EVENTS.NACK` flag, or assert the `CONTROLLER_HALT` Interrupt. This behavior is useful for protocols like Serial Camera Control Bus (SCCB). |
| 11 | `RCONT` | W | 0x0 | Read continue/stop: * `0` — Read Stop: The controller NACKs the last data byte. Use this if this is the last READ in a sequence. * `1` — Read Continue: The controller ACKs the last data byte. Use this if this is an intermediate READ in a sequence. |
| 10 | `READB` | W | 0x0 | Read/write: * `0` - Issue WRITE transaction * `1` - Issue READ transaction |
| 9 | `STOP` | W | 0x0 | Generate a STOP condition on the bus after sending the byte. |
| 8 | `START` | W | 0x0 | Generate a START condition on the bus before sending the byte. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | FBYTE | W WMOD | 0x0 | Writing to this register pushes an entry into the Controller TX FIFO. The meaning of this field depends on the value of the READB field: * READB = 0 - This field is the WRITE data byte. * READB = 1 - This field specifies the number of bytes to read from the target. Setting this field to 0 reads 256 bytes. |

### FIFO_CTRL

*Table 1964. i2c.FIFO_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | Reserved | - | - | Reserved |
| 8 | TXRST | W 1P | 0x0 | Target Mode control: writing 1 resets the Target TX FIFO. |
| 7 | ACQRST | W 1P | 0x0 | Target Mode control: writing 1 resets the Target RX FIFO. |
| 6:2 | Reserved | - | - | Reserved |
| 1 | FMTRST | W 1P | 0x0 | Controller Mode control: writing 1 resets the Controller TX FIFO. |
| 0 | RXRST | W 1P | 0x0 | Controller Mode control: writing 1 resets the Controller RX FIFO. |

### HOST_FIFO_CONFIG

*Table 1965. i2c.HOST_FIFO_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | - | - | Reserved |
| 27:16 | FMT_THRESH | RW | 0x0 | The FMT_THRESH interrupt remains asserted while the Controller TX FIFO level is less than this setting. |
| 15:12 | Reserved | - | - | Reserved |
| 11:0 | RX_THRESH | RW | 0x0 | The RX_THRESH interrupt remains asserted while the Controller RX FIFO level is greater than this setting. |

### TARGET_FIFO_CONFIG

*Table 1966. i2c.TARGET_FIFO_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | - | - | Reserved |
| 27:16 | ACQ_THRESH | RW | 0x0 | The ACQ_THRESH interrupt remains asserted while the Target RX FIFO level is greater than this setting. |
| 15:12 | Reserved | - | - | Reserved |
| 11:0 | TX_THRESH | RW | 0x0 | The TX_THRESH interrupt remains asserted while the Target TX FIFO level is less than this setting. |

### HOST_FIFO_STATUS

*Table 1967. i2c.HOST_FIFO_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | - | - | Reserved |
| 27:16 | RXLVL | R | - | Controller RX FIFO fill level. |
| 15:12 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 11:0 | FMTLVL | R | - | Controller TX FIFO fill level. |

### TARGET_FIFO_STATUS

*Table 1968. i2c.TARGET_FIFO_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | - | - | Reserved |
| 27:16 | ACQLVL | R | 0x0 | Target RX FIFO fill level. |
| 15:12 | Reserved | - | - | Reserved |
| 11:0 | TXLVL | R | 0x0 | Target TX FIFO fill level. |

### OVRD

*Table 1969. i2c.OVRD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | Reserved | - | - | Reserved |
| 2 | SDAVAL | RW | 0x0 | Global control: SDA override value: * 0 - Pull the SDA line low * 1 - Release the SDA line |
| 1 | SCLVAL | RW | 0x0 | Global control: SCL override value: * 0 - Pull the SCL line low * 1 - Release the SCL line |
| 0 | TXOVRDEN | RW | 0x0 | Global control: enables control of the SDA and SCL lines through the SDA_VAL and SCL_VAL fields, respectively. |

### VAL

*Table 1970. i2c.VAL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | SDA_RX | R | 0x0 | Global status: contains the last 16 SDA over-sampled values. LSB is most recent. |
| 15:0 | SCL_RX | R | 0x0 | Global status: contains the last 16 SCL over-sampled values. LSB is most recent. |

### TIMING0

*Table 1971. i2c.TIMING0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | Reserved | - | - | Reserved |
| 28:16 | TLOW | RW | 0x0 | Global control: specifies the LOW period of SCL (t_LOW) in system clock cycles. Must be  2. See Table 11 in the I2C Specification for more details. This field is sized to meet the I2C Standard-mode (100 kHz) minimum t_LOW = 4.7 µs requirement, assuming a 1 GHz system clock. |
| 15:13 | Reserved | - | - | Reserved |
| 12:0 | THIGH | RW | 0x0 | Global control: specifies the HIGH period of SCL (t_HIGH) in system clock cycles. Must be  2. See Table 11 in the I²C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s minimum t_HIGH = 4.0 µs requirement, assuming a 1 GHz system clock. |

### TIMING1

*Table 1972. i2c.TIMING1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | `Reserved` | - | - | Reserved |
| 24:16 | `T_F` | RW | 0x0 | Global control: specifies the fall time of SDA and SCL (`t_f`) in system clock cycles. The fall time is measured from 70% to 30% of the signal swing. See Table 11 in the I2C Specification for more details. This field is sized to meet I2C Standard-mode (100 kHz)'s maximum `t_f = 300 ns` requirement, assuming a 1 GHz system clock. |
| 15:10 | `Reserved` | - | - | Reserved |
| 9:0 | `T_R` | RW | 0x0 | Global control: specifies the rise time of SDA and SCL (`t_r`) in system clock cycles. The rise time is measured from 30% to 70% of the signal swing. See Table 11 in the I2C Specification for more details. This field is sized to meet I2C Standard-mode (100 kHz)'s maximum `t_r = 1000 ns` requirement, assuming a 1 GHz system clock. |

### TIMING2

*Table 1973. i2c.TIMING2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `THD_STA` | RW | 0x0 | Global control: specifies the setup time for a (repeated) START condition (`t_HD;STA`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s minimum `t_HD;STA = 4.0 µs` requirement, assuming a 1 GHz system clock. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12:0 | `TSU_STA` | RW | 0x0 | Global control: specifies the setup time for a repeated START condition (`t_SU;STA`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s minimum `t_SU;STA = 4.7 µs` requirement, assuming a 1 GHz system clock. |

### TIMING3

*Table 1974. i2c.TIMING3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `THD_DAT` | RW | 0x0 | Global control: specifies the data and (N)ACK bits hold time (`t_HD;DAT`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s `t_HD;DAT = 5.0 µs` minimum requirement, assuming a 1 GHz system clock. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8:0 | `TSU_DAT` | RW | 0x0 | Global control: specifies the data setup time (`t_SU;DAT`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s `t_SU;DAT = 250 ns` minimum requirement, assuming a 1 GHz system clock. |

### TIMING4

*Table 1975. i2c.TIMING4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `T_BUF` | RW | 0x0 | Global control: specifies the time between a STOP and START condition (`t_BUF`) in system clock cycles. See Table 11 in the I²C Specification for details. This field is sized to meet I2C Standard-mode (100kHz)'s `t_BUF = 4.7 µs` minimum requirement, assuming a 1 GHz system clock. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12:0 | `TSU_STO` | RW | 0x0 | Global control: specifies the setup time for a STOP condition (`t_SU;STO`) in system clock cycles. See Table 11 in the I²C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s `t_SU;STO = 4.0 µs` minimum requirement, assuming a 1 GHz system clock. |

### TIMEOUT_CTRL

*Table 1976. i2c.TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Timeout Enable. |
| 30 | `MODE` | RW | 0x0 | Global control: timeout mode. * `0` - Stretch Timeout (Controller Mode only). If the target stretches the clock for more time than `TIMEOUT_CTRL.VAL`, the `STRETCH_TIMEOUT` interrupt will be asserted. * `1` - Bus Timeout. If SCL is LOW for more time than `TIMEOUT_CTRL.VAL`: * In Controller Mode, the `CONTROLLER_EVENTS.BUS_TIMEOUT` flag will be asserted, triggering the `CONTROLLER_HALT` interrupt. * In Target Mode, the `TARGET_EVENTS.BUS_TIMEOUT` flag will be set, triggering the `TX_STRETCH` interrupt. |
| 29:0 | `VAL` | RW | 0x0 | Global control: Specifies the timeout value in system clock cycles. The meaning of this field depends on `MODE`. |

### TARGET_ID

*Table 1977. i2c.TARGET_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:21 | `MASK1` | RW | 0x0 | ADDRESS1 mask. |
| 20:14 | `ADDRESS1` | RW | 0x0 | Target Address 1. This target responds if the 7-bit address matches `ADDRESS1 & MASK1`. `MASK1 = 0x0` disables this address. |
| 13:7 | `MASK0` | RW | 0x0 | ADDRESS0 mask. |
| 6:0 | `ADDRESS0` | RW | 0x0 | Target Address 0. This target responds if the 7-bit address matches `ADDRESS0 & MASK0`. `MASK0 = 0x0` disables this address. |

### ACQDATA

*Table 1978. i2c.ACQDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 10:8 | `SIGNAL` | R | 0x0 | This field indicates if this FIFO entry represents/is associated with control signal(s): * `0x0` - The entry is an ordinary data byte that has been ACKed. * `0x1` - The entry is an address byte preceded by a START. * `0x2` - The entry is a STOP condition following ACKed data bytes. * `0x3` - The entry is an address byte preceded by a repeated START. * `0x4` - The entry is a NACKed data byte. * `0x5` - The entry is an address byte preceded by a (repeated) START. However, the data bytes that followed this address byte were NACKed. * `0x6` - Error. A transaction preceding ended abnormally, for example, due to an unexpected STOP condition, a Bus or Stretch Timeout, a software NACK, or a lost arbitration.<br><br>If the FIFO does not have enough space to record a complete transaction, an Error (`0x6`) entry may appear alone. |
| 7:0 | `ABYTE` | R RMOD | 0x0 | Reading this register pops the ACQ FIFO. The meaning of this field depends on `SIGNAL`: * `SIGNAL = 0x0`, `0x1`, `0x3`, `0x4`, or `0x5` - This field contains an address or data byte sent by the controller. * `SIGNAL = 0x2`, `0x6` - This field is meaningless. |

### TXDATA

*Table 1979. i2c.TXDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | W WMOD | 0x0 | Writing to this register pushes data into the Target TX FIFO. The controller reads data from this FIFO during a READ transaction. |

### HOST_TIMEOUT_CTRL

*Table 1980. i2c.HOST_TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30:0 | `VAL` | RW | 0x0 | Target/Monitor Mode control: controller clock generation timeout value specified in system clock cycles. * Target Mode - If the controller stops generating the clock for more time than this setting, this target asserts the Controller Timeout Interrupt. * Monitor Mode - this field is required to be nonzero for the Bus Monitor to transition out of the initial Busy state. Set this field to `0x0` to disable this behavior. |

### TARGET_TIMEOUT_CTRL

*Table 1981. i2c.TARGET_TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Enable Target Timeout. |
| 30:0 | `VAL` | RW | 0x0 | When this target has stretched the clock for more time than this setting, this target will NACK incoming data bytes or release the SDA line for outgoing data bytes. The count is cumulative over an entire transaction. In other words, this is SMBus's cumulative target clock extension time.<br><br>The behavior for the address byte is configurable via `CTRL.NACK_ADDR_AFTER_TIMEOUT`. |

### TARGET_NACK_COUNT

*Table 1982. i2c.TARGET_NACK_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `TARGET_NACK_COUNT` | RW RC | 0x0 | Indicates the number of transactions NACKed by this target since the last read of this register, saturating at 255. This field can be used to track how many transactions were missed when the Target RX FIFO is full. |

### TARGET_ACK_CTRL

*Table 1983. i2c.TARGET_ACK_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `NACK` | W 1P | 0x0 | When `STATUS.ACK_CTRL_STRETCH = 1`, writing `1` to this field causes this target to NACK all bytes of the transaction. Effective only when `CTRL.ACK_CTRL_EN = 1`. |
| 30:9 | `Reserved` | - | - | Reserved |
| 8:0 | `NBYTES` | RW | 0x0 | When `STATUS.ACK_CTRL_STRETCH = 1`, writing to this register specifies the number of bytes this target should ACK. The count decrements per byte ACKed. Effective only when `CTRL.ACK_CTRL_EN = 1`. See `CTRL.ACK_CTRL_EN` for details.<br><br>This control field is used to implement SMBus's mid-transfer (N)ACK responses. |

### ACQ_FIFO_NEXT_DATA

*Table 1984. i2c.ACQ_FIFO_NEXT_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R | 0x0 | This field contains the next byte to be pushed into the Target RX FIFO, allowing software to decide whether to accept or reject it. Valid only when `STATUS.ACK_CTRL_STRETCH = 1`. See `CTRL.ACK_CTRL_EN` for details. |

### HOST_NACK_HANDLER_TIMEOUT

*Table 1985. i2c.HOST_NACK_HANDLER_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Enables Controller NACK Timeout. |
| 30:0 | `VAL` | RW | 0x0 | Timeout value (specified in system clock cycles) for this controller Mode to automatically STOP a transaction when `CONTROLLER_EVENTS.NACK` is set. |

### CONTROLLER_EVENTS

*Table 1986. i2c.CONTROLLER_EVENTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `ARBITRATION_LOST` | RW1C | 0x0 | Indicates that the controller has halted due to it losing an arbitration against another controller. This bit triggers the `CONTROLLER_HALT` interrupt. Writing `1` clears this bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | BUS_TIMEOUT | RW1C | 0x0 | Indicates that this controller has halted due to a Bus Timeout. See TIMEOUT_CTRL for details. This bit triggers the CONTROLLER_HALT interrupt. Writing 1 clears this bit. |
| 1 | UNHANDLED_NACK_TIMEOUT | RW1C | 0x0 | Indicates that this controller has halted due to a Controller NACK Timeout. See HOST_NACK_HANDLER_TIMEOUT for details. This bit triggers the CONTROLLER_HALT interrupt. Writing 1 clears this bit. |
| 0 | NACK | RW1C | 0x0 | Indicates that this controller has halted due to an unexpected NACK sent by the target. This behavior can be disabled by writing 1 to FDATA.NAKOK. This bit triggers the CONTROLLER_HALT interrupt. Writing 1 clears this bit. |

#### TARGET_EVENTS

*Table 1987. i2c.TARGET_EVENTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | STOP_DETECT | RW1C | 0x0 | Stop Detect Flag (Target Mode only). Set to 1 by hardware when a STOP is detected while CTRL.ENABLETARGET = 1. Cleared when software writes 1. |
| 3 | START_DETECT | RW1C | 0x0 | Start Detect Flag (Target Mode only). Set to 1 by hardware when a START (or repeated START) is detected while CTRL.ENABLETARGET = 1. Cleared when software writes 1. |
| 2 | ARBITRATION_LOST | RW1C | 0x0 | Indicates that a controller has lost arbitration, causing a READ transaction to end. This bit triggers the TX_STRETCH interrupt. Writing 1 clears this bit. |
| 1 | BUS_TIMEOUT | RW1C | 0x0 | Indicates that this target has halted due a Bus Timeout terminating a READ transaction. See TIMEOUT_CTRL for details. This bit triggers the TX_STRETCH interrupt. Writing 1 clears this bit. |
| 0 | TX_PENDING | RW1C | 0x0 | Indicates that the target is stretching the clock due to receiving a READ address byte and waiting for software to confirm the release of the Target TX FIFO data. Valid only if CTRL.TX_STRETCH_CTRL_EN = 1. See CTRL.TX_STRETCH_CTRL_EN for details. This bit triggers the TX_STRETCH interrupt. Writing 1 clears this bit. |

#### SMBUS_STATUS

*Table 1988. i2c.SMBUS_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | SMBALERT | R | 0x0 | Controller Mode status: indicates that the smbalert_ni input is asserted. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | SMBSUS | R | 0x0 | Target Mode status: indicates that the smbsus_ni input is asserted. |

### I2C Control Address Map

Per-instance pad-function select, SMBus pad direction, and SMBus alert enable.

**NOTE**

**Register Arrays:**

- **I2C_CTRL[3]**: 3 registers, base 0x0, stride 0x4

*Table 1989. I2C Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x8 | I2C_CTRL[3] | RW | Control Register |

### I2C_CTRL[3]

*Table 1990. i2c_ctrl.I2C_CTRL[3] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `SMBUS_EN` | RW | 0x0 | SMBus Enable. Gates the SMBus Alert signal. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `I2C_CONTROLLER_MODE_EN` | RW | 0x0 | SMBus Pad Direction. When set, the SMBus alert pad is an input and the SMBus suspend pad is an output, as for an SMBus host. When clear, the alert pad is an output and the suspend pad is an input, as for an SMBus target. SCL and SDA are unaffected. Program it to agree with the core's CTRL.ENABLEHOST and CTRL.ENABLETARGET setting. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `I2C_EN` | RW | 0x0 | I2C Pad Function Select. When set, the integration routes this instance's SCL, SDA, SMBus alert and SMBus suspend signals to their pads. Does not enable the I2C core; CTRL.ENABLEHOST and CTRL.ENABLETARGET select the core's operating mode. |

### I2C Wrapper Address Map

Wrapper address map instantiating the per-controller I2C register blocks and shared control.

**NOTE**

**Register Arrays:**

- **i2c[3].INTR_STATE**: 3 registers, base 0x0, stride 0x200
- **i2c[3].INTR_ENABLE**: 3 registers, base 0x4, stride 0x200
- **i2c[3].INTR_TEST**: 3 registers, base 0x8, stride 0x200
- **i2c[3].SMBUS_CTRL**: 3 registers, base 0xC, stride 0x200
- **i2c[3].CTRL**: 3 registers, base 0x10, stride 0x200
- **i2c[3].STATUS**: 3 registers, base 0x14, stride 0x200
- **i2c[3].RDATA**: 3 registers, base 0x18, stride 0x200
- **i2c[3].FDATA**: 3 registers, base 0x1C, stride 0x200
- **i2c[3].FIFO_CTRL**: 3 registers, base 0x20, stride 0x200
- **i2c[3].HOST_FIFO_CONFIG**: 3 registers, base 0x24, stride 0x200
- **i2c[3].TARGET_FIFO_CONFIG**: 3 registers, base 0x28, stride 0x200
- **i2c[3].HOST_FIFO_STATUS**: 3 registers, base 0x2C, stride 0x200
- **i2c[3].TARGET_FIFO_STATUS**: 3 registers, base 0x30, stride 0x200
- **i2c[3].OVRD**: 3 registers, base 0x34, stride 0x200
- **i2c[3].VAL**: 3 registers, base 0x38, stride 0x200
- **i2c[3].TIMING0**: 3 registers, base 0x3C, stride 0x200
- **i2c[3].TIMING1**: 3 registers, base 0x40, stride 0x200

- **i2c[3].TIMING2**: 3 registers, base 0x44, stride 0x200
- **i2c[3].TIMING3**: 3 registers, base 0x48, stride 0x200
- **i2c[3].TIMING4**: 3 registers, base 0x4C, stride 0x200
- **i2c[3].TIMEOUT_CTRL**: 3 registers, base 0x50, stride 0x200
- **i2c[3].TARGET_ID**: 3 registers, base 0x54, stride 0x200
- **i2c[3].ACQDATA**: 3 registers, base 0x58, stride 0x200
- **i2c[3].TXDATA**: 3 registers, base 0x5C, stride 0x200
- **i2c[3].HOST_TIMEOUT_CTRL**: 3 registers, base 0x60, stride 0x200
- **i2c[3].TARGET_TIMEOUT_CTRL**: 3 registers, base 0x64, stride 0x200
- **i2c[3].TARGET_NACK_COUNT**: 3 registers, base 0x68, stride 0x200
- **i2c[3].TARGET_ACK_CTRL**: 3 registers, base 0x6C, stride 0x200
- **i2c[3].ACQ_FIFO_NEXT_DATA**: 3 registers, base 0x70, stride 0x200
- **i2c[3].HOST_NACK_HANDLER_TIMEOUT**: 3 registers, base 0x74, stride 0x200
- **i2c[3].CONTROLLER_EVENTS**: 3 registers, base 0x78, stride 0x200
- **i2c[3].TARGET_EVENTS**: 3 registers, base 0x7C, stride 0x200
- **i2c[3].SMBUS_STATUS**: 3 registers, base 0x80, stride 0x200
- **I2C_CTRL[3]**: 3 registers, base 0xE00, stride 0x4

*Table 1991. I2C Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x400 | i2c[3].INTR_STATE | RW | Interrupt Status Register |
| 0x4 - 0x404 | i2c[3].INTR_ENABLE | RW | Interrupt Enable Register |
| 0x8 - 0x408 | i2c[3].INTR_TEST | RW | Interrupt Test Register |
| 0xC - 0x40C | i2c[3].SMBUS_CTRL | RW | SMBus Control Register |
| 0x10 - 0x410 | i2c[3].CTRL | RW | Control Register |
| 0x14 - 0x414 | i2c[3].STATUS | R | Status Register |
| 0x18 - 0x418 | i2c[3].RDATA | R | Controller RX FIFO Access Register (Controller Mode only) |
| 0x1C - 0x41C | i2c[3].FDATA | W | Controller TX FIFO Access Register (Controller Mode only) |
| 0x20 - 0x420 | i2c[3].FIFO_CTRL | W | FIFO Control Register |
| 0x24 - 0x424 | i2c[3].HOST_FIFO_CONFIG | RW | Controller FIFO Configuration Register (Controller Mode only) |
| 0x28 - 0x428 | i2c[3].TARGET_FIFO_CONFIG | RW | Target FIFOs Configuration Register (Target Mode only) |
| 0x2C - 0x42C | i2c[3].HOST_FIFO_STATUS | R | Controller FIFOs Status Register (Controller Mode only) |
| 0x30 - 0x430 | i2c[3].TARGET_FIFO_STATUS | R | Target FIFOs Status Register (Target Mode only) |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x34 - 0x434 | i2c[3].OVRD | RW | Override Control Register |
| 0x38 - 0x438 | i2c[3].VAL | R | Bus Oversampled Values Register |
| 0x3C - 0x43C | i2c[3].TIMING0 | RW | SCL LOW and HIGH Periods Register |
| 0x40 - 0x440 | i2c[3].TIMING1 | RW | Bus Rise and Fall Times Register |
| 0x44 - 0x444 | i2c[3].TIMING2 | RW | START Condition Setup and Hold Times Register |
| 0x48 - 0x448 | i2c[3].TIMING3 | RW | Data Setup and Hold Times Register |
| 0x4C - 0x44C | i2c[3].TIMING4 | RW | STOP Condition Setup Time and Bus Free Time Register |
| 0x50 - 0x450 | i2c[3].TIMEOUT_CTRL | RW | Timeout Control Register |
| 0x54 - 0x454 | i2c[3].TARGET_ID | RW | Target ID Register (Target Mode only) |
| 0x58 - 0x458 | i2c[3].ACQDATA | R | Target RX FIFO Access Register (Target Mode only) |
| 0x5C - 0x45C | i2c[3].TXDATA | W | Target TX FIFO Access Register (Target Mode only). |
| 0x60 - 0x460 | i2c[3].HOST_TIMEOUT_CTRL | RW | Controller Timeout Control Register |
| 0x64 - 0x464 | i2c[3].TARGET_TIMEOUT_CTRL | RW | Target Timeout Control Register (Target Mode only) |
| 0x68 - 0x468 | i2c[3].TARGET_NACK_COUNT | RW | Target NACK Count Register (Target Mode only) |
| 0x6C - 0x46C | i2c[3].TARGET_ACK_CTRL | RW | Target ACK Control Register (Target Mode only) |
| 0x70 - 0x470 | i2c[3].ACQ_FIFO_NEXT_DATA | R | Target RX FIFO Next Byte Register (Target Mode only) |
| 0x74 - 0x474 | i2c[3].HOST_NACK_HANDLER_TIMEOUT | RW | Controller NACK Timeout Register (Controller Mode only) |
| 0x78 - 0x478 | i2c[3].CONTROLLER_EVENTS | RW | Controller Events Register (Controller Mode only) |
| 0x7C - 0x47C | i2c[3].TARGET_EVENTS | RW | Target Events Register (Target Mode) |
| 0x80 - 0x480 | i2c[3].SMBUS_STATUS | R | SMBus Status Register |
| 0xE00 - 0xE08 | I2C_CTRL[3] | RW | Control Register |

### i2c[3].INTR_STATE

*Table 1992. i2c_wrap.i2c[3].INTR_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19 | `TARGET_RX_FIFO_ERROR` | RW1C | 0x0 | Target Mode interrupt: asserted when the Target RX FIFO has a parity error. Write 1 to clear. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 18 | TARGET_TX_FIFO_ERROR | RW1C | 0x0 | Target Mode interrupt: asserted when the Target TX FIFO has a parity error. Write 1 to clear. |
| 17 | CONTROLLER_RX_FIFO_ERROR | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller RX FIFO has a parity error. Write 1 to clear. |
| 16 | CONTROLLER_TX_FIFO_ERROR | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller TX FIFO has a parity error. Write 1 to clear. |
| 15 | SMBALERT | RW1C | 0x0 | Controller Mode interrupt: asserted when smbalert_ni is asserted. Write 1 to clear. |
| 14 | HOST_TIMEOUT | RW1C | 0x0 | Target Mode interrupt: asserted when the controller stops generating the clock longer than HOST_TIMEOUT_CTRL.VAL. Write 1 to clear. |
| 13 | UNEXP_STOP | RW1C | 0x0 | Target Mode interrupt: asserted when the controller sends this target a STOP before this target NACKs. Write 1 to clear. |
| 12 | ACQ_STRETCH | R | 0x0 | Target Mode interrupt: remains asserted while the target is stretching the clock because 1) the Target RX FIFO is full or 2) the TARGET_ACK_CTRL.NBYTES count has reached 0 (only if CTRL.ACK_CTRL_EN = 1). |
| 11 | TX_THRESHOLD | R | 0x0 | Target Mode interrupt: remains asserted while the Target TX FIFO level is less than TARGET_FIFO_CONFIG.TX_THRESH. |
| 10 | TX_STRETCH | R | 0x0 | Target Mode interrupt: remains asserted while this target is stretching the clock or has halted. The flags in the TARGET_EVENTS register explain the reason(s) for clock stretching/halting. Clearing the TARGET_EVENTS flags clears this interrupt. |
| 9 | CMD_COMPLETE | RW1C | 0x0 | Controller/Target Mode interrupt: asserted when this/the controller finishes generating a STOP or repeated START. Write 1 to clear. |
| 8 | SDA_UNSTABLE | RW1C | 0x0 | Controller Mode interrupt: asserted when the target fails to keep SDA stable during a transmission. Write 1 to clear. |
| 7 | STRETCH_TIMEOUT | RW1C | 0x0 | Controller Mode interrupt: asserted when the target stretches the clock longer than TIMEOUT_CTRL.VAL (valid only when TIMEOUT_CTRL.MODE = 0). Write 1 to clear. |
| 6 | SDA_INTERFERENCE | RW1C | 0x0 | Controller Mode interrupt: asserted when SDA is unexpectedly pulled LOW by another controller. Write 1 to clear. |
| 5 | SCL_INTERFERENCE | RW1C | 0x0 | Controller Mode interrupt: asserted when SCL is unexpectedly pulled LOW by another controller. Write 1 to clear. |
| 4 | CONTROLLER_HALT | R | 0x0 | Controller Mode interrupt: remains asserted while this controller halts. The flags in the CONTROLLER_EVENTS register explain the reason(s) for the halting. Clearing the CONTROLLER_EVENTS flags clears this interrupt. |
| 3 | RX_OVERFLOW | RW1C | 0x0 | Controller Mode interrupt: asserted when the Controller RX FIFO overflows. Write 1 to clear. |
| 2 | ACQ_THRESHOLD | R | 0x0 | Target Mode interrupt: remains asserted while the Target RX FIFO level is greater than TARGET_FIFO_CONFIG.ACQ_THRESH. |
| 1 | RX_THRESHOLD | R | 0x0 | Controller Mode interrupt: remains asserted while the Controller RX FIFO level is greater than HOST_FIFO_CONFIG.RX_THRESH. |
| 0 | FMT_THRESHOLD | R | 0x0 | Controller Mode interrupt: remains asserted while the Controller TX FIFO level is less than HOST_FIFO_CONFIG.FMT_THRESH. |

### i2c[3].INTR_ENABLE

*Table 1993. i2c_wrap.i2c[3].INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | TARGET_RX_FIFO_ERROR | RW | 0x0 | Enables the TARGET_RX_FIFO_ERROR interrupt. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 18 | TARGET_TX_FIFO_ERROR | RW | 0x0 | Enables the TARGET_TX_FIFO_ERROR interrupt. |
| 17 | CONTROLLER_RX_FIFO_ERROR | RW | 0x0 | Enables the CONTROLLER_RX_FIFO_ERROR interrupt. |
| 16 | CONTROLLER_TX_FIFO_ERROR | RW | 0x0 | Enables the CONTROLLER_TX_FIFO_ERROR interrupt. |
| 15 | SMBALERT | RW | 0x0 | Enables the SMBALERT interrupt. |
| 14 | HOST_TIMEOUT | RW | 0x0 | Enables the HOST_TIMEOUT interrupt. |
| 13 | UNEXP_STOP | RW | 0x0 | Enables the UNEXP_STOP interrupt. |
| 12 | ACQ_STRETCH | RW | 0x0 | Enables the ACQ_STRETCH interrupt. |
| 11 | TX_THRESHOLD | RW | 0x0 | Enables the TX_THRESHOLD interrupt. |
| 10 | TX_STRETCH | RW | 0x0 | Enables the TX_STRETCH interrupt. |
| 9 | CMD_COMPLETE | RW | 0x0 | Enables the CMD_COMPLETE interrupt. |
| 8 | SDA_UNSTABLE | RW | 0x0 | Enables the SDA_UNSTABLE interrupt. |
| 7 | STRETCH_TIMEOUT | RW | 0x0 | Enables the STRETCH_TIMEOUT interrupt. |
| 6 | SDA_INTERFERENCE | RW | 0x0 | Enables the SDA_INTERFERENCE interrupt. |
| 5 | SCL_INTERFERENCE | RW | 0x0 | Enables the SCL_INTERFERENCE interrupt. |
| 4 | CONTROLLER_HALT | RW | 0x0 | Enables the CONTROLLER_HALT interrupt. |
| 3 | RX_OVERFLOW | RW | 0x0 | Enables the RX_OVERFLOW interrupt. |
| 2 | ACQ_THRESHOLD | RW | 0x0 | Enables the ACQ_THRESHOLD interrupt. |
| 1 | RX_THRESHOLD | RW | 0x0 | Enables the RX_THRESHOLD interrupt. |
| 0 | FMT_THRESHOLD | RW | 0x0 | Enables the FMT_THRESHOLD interrupt. |

### i2c[3].INTR_TEST

*Table 1994. i2c_wrap.i2c[3].INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | TARGET_RX_FIFO_ERROR | W 1P | 0x0 | Writing 1 forces the TARGET_RX_FIFO_ERROR interrupt. |
| 18 | TARGET_TX_FIFO_ERROR | W 1P | 0x0 | Writing 1 forces the TARGET_TX_FIFO_ERROR interrupt. |
| 17 | CONTROLLER_RX_FIFO_ERROR | W 1P | 0x0 | Writing 1 forces the CONTROLLER_RX_FIFO_ERROR interrupt. |
| 16 | CONTROLLER_TX_FIFO_ERROR | W 1P | 0x0 | Writing 1 forces the CONTROLLER_TX_FIFO_ERROR interrupt. |
| 15 | SMBALERT | W 1P | 0x0 | Writing 1 forces the SMBALERT interrupt. |
| 14 | HOST_TIMEOUT | W 1P | 0x0 | Writing 1 forces the HOST_TIMEOUT interrupt. |
| 13 | UNEXP_STOP | W 1P | 0x0 | Writing 1 forces the UNEXP_STOP interrupt. |
| 12 | ACQ_STRETCH | RW | 0x0 | Writing 1 forces the ACQ_STRETCH interrupt. Writing 0 releases it. |
| 11 | TX_THRESHOLD | RW | 0x0 | Writing 1 forces the TX_THRESHOLD interrupt. Writing 0 releases it. |
| 10 | TX_STRETCH | RW | 0x0 | Writing 1 forces the TX_STRETCH interrupt. Writing a 0 releases it. |
| 9 | CMD_COMPLETE | W 1P | 0x0 | Writing 1 forces the CMD_COMPLETE interrupt. |
| 8 | SDA_UNSTABLE | W 1P | 0x0 | Writing 1 forces the SDA_UNSTABLE interrupt. |
| 7 | STRETCH_TIMEOUT | W 1P | 0x0 | Writing 1 forces the STRETCH_TIMEOUT interrupt. |
| 6 | SDA_INTERFERENCE | W 1P | 0x0 | Writing 1 forces the SDA_INTERFERENCE interrupt. |
| 5 | SCL_INTERFERENCE | W 1P | 0x0 | Writing 1 forces the SCL_INTERFERENCE interrupt. |
| 4 | CONTROLLER_HALT | RW | 0x0 | Writing 1 forces the CONTROLLER_HALT interrupt. Writing 0 releases it. |
| 3 | RX_OVERFLOW | W 1P | 0x0 | Writing 1 forces the RX_OVERFLOW interrupt. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | `ACQ_THRESHOLD` | RW | 0x0 | Writing `1` forces the `ACQ_THRESHOLD` interrupt. Writing `0` releases it. |
| 1 | `RX_THRESHOLD` | RW | 0x0 | Writing `1` forces the `RX_THRESHOLD` interrupt. Writing `0` releases it. |
| 0 | `FMT_THRESHOLD` | RW | 0x0 | Writing `1` forces the `FMT_THRESHOLD` interrupt. Writing `0` releases it. |

### i2c[3].SMBUS_CTRL

*Table 1995. i2c_wrap.i2c[3].SMBUS_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `SMBALERT` | RW | 0x0 | Target Mode control: asserts the `smbalert_no` output. This bit clears itself when the controller addresses this target. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `SMBSUS` | RW | 0x0 | Controller Mode control: asserts the `smbsus_no` output. |

### i2c[3].CTRL

*Table 1996. i2c_wrap.i2c[3].CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `ACQ_START_STOP_EN` | RW | 0x0 | ACQ FIFO Start/Stop Enable (Target Mode only): * `0` - Start/Stop symbols are NOT written into the ACQ FIFO. The firmware should rely on `TARGET_EVENTS.START_DETECT` and `TARGET_EVENTS.STOP_DETECT` flags to detect Start/Stop events. * `1` - Start/Stop symbols ARE written into the ACQ FIFO (legacy behavior). When a START (or repeated START) is detected, an `ACQ_START` or `ACQ_RESTART` entry is written. When a STOP is detected, an `ACQ_STOP` or `ACQ_NACK_STOP` entry is written. |
| 6 | `TX_STRETCH_CTRL_EN` | RW | 0x0 | Target mode control: enables the Software TX Stretch Control Mechanism. If this bit is: * `0` - Automatic TX Stretch: When this target receives a READ address byte, it only stretches the clock if the TX FIFO is empty; otherwise, it pops the FIFO and transmits the data byte. The target never sets the `TARGET_EVENTS` register flags in this mode. * `1` - Software TX Stretch Mode: When this target receives a READ address, it always stretches the clock and sets the `TARGET_EVENTS.TX_PENDING` flag. The software can: 1. confirm the release and transmission of the TX FIFO data by writing a 1 to clear the `TARGET_EVENTS.TX_PENDING` flag; or 2. reset the TX FIFO by writing 1 to `FIFO_CTRL.TXRST` and load in new data via the `TXDATA` register—useful whenthe READ address is targeting a different function of the target. In this mode, the target always sets the `TARGET_EVENTS` register flags. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 5 | MULTI_CONTROLLER_MONITOR_EN | RW | 0x0 | Global control: enables the Bus Monitor. Set this bit to 1 only in a multi-controller environment.<br><br>If a 0→`1` transition happens while ENABLEHOST and ENABLETARGET are both 0, the Bus Monitor will enable and begin in the 'Bus Busy' state. To transition to a 'Bus Free' state, HOST_TIMEOUT_CTRL must be nonzero so the Bus Monitor may count out idle cycles to confirm the freedom to transmit. In addition, the Bus Monitor will track whether the bus is free based on the enabled timeouts and detected STOP symbols. For Multi-Controller Mode, ensure MULTI_CONTROLLER_MONITOR_EN becomes 1 no later than ENABLEHOST or ENABLETARGET. This bit can be set at the same time as either or both of the other two, though.<br><br>Note that if MULTI_CONTROLLER_MONITOR_EN is set after ENABLEHOST or ENABLETARGET, the Bus Monitor will begin in the 'Bus Free' state instead. This would violate the proper protocol for a controller to join a multi-controller environment. However, if this controller is known to be the first to join, this ordering will enable skipping the idle wait.<br><br>When 0, the bus monitor will report that the bus is always free, so the Controller FSM is never blocked from transmitting. |
| 4 | ACK_CTRL_EN | RW | 0x0 | Target Mode control: enables the Software ACK Control Mechanism. If this bit is: * 0 - This target ACKs a data byte whenever the ACQ FIFO has space. * 1 - This target ACKs the first TARGET_ACK_CTRL.NBYTES bytes. If another byte arrives, this target stretches the clock and awaits software intervention (and asserts STATUS.ACK_CTRL_STRETCH). The software can 1. accept the new byte(s) by reloading the TARGET_ACK_CTRL.NBYTES counter; or 2. reject the new byte(s) by writing 1 to TARGET_ACK_CTRL.NACK (useful for SMBus). |
| 3 | NACK_ADDR_AFTER_TIMEOUT | RW | 0x0 | Target Mode control: NACK address after timeout. If this bit is: * 0 - This target ACKs the address byte even if a Stretch Timeout occurs (useful for SMBus). * 1 - This target NACKs the address byte when a Stretch Timeout occurs. |
| 2 | LLPBK | RW | 0x0 | Global control: enables line loop-back. In Controller Mode, the internal logic sees the smbalert_ni input as deasserted. In Target Mode, this target sends all received SDA data back out, and the internal logic sees received data as all 1's. |
| 1 | ENABLETARGET | RW | 0x0 | Global control: configures this device's operating mode. * ENABLEHOST = 1 and ENABLETARGET = 0 — Controller Mode * ENABLEHOST = 0 and ENABLETARGET = 1 — Target Mode * ENABLEHOST = 1 and ENABLETARGET = 1 — Hybrid Mode * ENABLEHOST = 0 and ENABLETARGET = 0 — Monitor Mode |
| 0 | ENABLEHOST | RW | 0x0 | Global control: configures this device's operating mode. * ENABLEHOST = 1 and ENABLETARGET = 0 — Controller Mode * ENABLEHOST = 0 and ENABLETARGET = 1 — Target Mode * ENABLEHOST = 1 and ENABLETARGET = 1 — Hybrid Mode * ENABLEHOST = 0 and ENABLETARGET = 0 — Monitor Mode |

### i2c[3].STATUS

*Table 1997. i2c_wrap.i2c[3].STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | Reserved | - | - | Reserved |
| 10 | ACK_CTRL_STRETCH | R | 0x0 | Target Mode status: indicates that this target is stretching the clock due to the Software ACK Control Mechanism. See CTRL.ACK_CTRL_EN for details. |
| 9 | ACQEMPTY | R | 0x1 | Target Mode status: Target RX FIFO empty. |
| 8 | TXEMPTY | R | 0x1 | Target Mode status: Target TX FIFO empty. |
| 7 | ACQFULL | R | 0x0 | Target Mode status: Target RX FIFO full. |
| 6 | TXFULL | R | 0x0 | Target Mode status: Target TX FIFO full. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 5 | RXEMPTY | R | 0x1 | Controller Mode status: Controller RX FIFO empty. |
| 4 | TARGETIDLE | R | 0x1 | Target Mode status: Target FSM idle. |
| 3 | HOSTIDLE | R | 0x1 | Controller Mode status: Controller FSM idle. |
| 2 | FMTEMPTY | R | 0x1 | Controller Mode status: Controller TX FIFO Empty. |
| 1 | RXFULL | R | 0x0 | Controller Mode status: Controller RX FIFO Full. |
| 0 | FMTFULL | R | 0x0 | Controller Mode status: Controller TX FIFO Full. |

### i2c[3].RDATA

*Table 1998. i2c_wrap.i2c[3].RDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | DATA | R RMOD | 0x0 | Reading this register pops the Controller RX FIFO. |

### i2c[3].FDATA

*Table 1999. i2c_wrap.i2c[3].FDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | Reserved | - | - | Reserved |
| 12 | NAKOK | W | 0x0 | NACK OK. When set, this controller will not care if this byte is NACKed. It will not halt, set the CONTROLLER_EVENTS.NACK flag, or assert the CONTROLLER_HALT Interrupt. This behavior is useful for protocols like Serial Camera Control Bus (SCCB). |
| 11 | RCONT | W | 0x0 | Read continue/stop: * 0 — Read Stop: The controller NACKs the last data byte. Use this if this is the last READ in a sequence. * 1 — Read Continue: The controller ACKs the last data byte. Use this if this is an intermediate READ in a sequence. |
| 10 | READB | W | 0x0 | Read/write: * 0 - Issue WRITE transaction * 1 - Issue READ transaction |
| 9 | STOP | W | 0x0 | Generate a STOP condition on the bus after sending the byte. |
| 8 | START | W | 0x0 | Generate a START condition on the bus before sending the byte. |
| 7:0 | FBYTE | W WMOD | 0x0 | Writing to this register pushes an entry into the Controller TX FIFO. The meaning of this field depends on the value of the READB field: * READB = 0 - This field is the WRITE data byte. * READB = 1 - This field specifies the number of bytes to read from the target. Setting this field to 0 reads 256 bytes. |

### i2c[3].FIFO_CTRL

*Table 2000. i2c_wrap.i2c[3].FIFO_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | Reserved | - | - | Reserved |
| 8 | TXRST | W 1P | 0x0 | Target Mode control: writing 1 resets the Target TX FIFO. |
| 7 | ACQRST | W 1P | 0x0 | Target Mode control: writing 1 resets the Target RX FIFO. |
| 6:2 | Reserved | - | - | Reserved |
| 1 | FMTRST | W 1P | 0x0 | Controller Mode control: writing 1 resets the Controller TX FIFO. |
| 0 | RXRST | W 1P | 0x0 | Controller Mode control: writing 1 resets the Controller RX FIFO. |

### i2c[3].HOST_FIFO_CONFIG

*Table 2001. i2c_wrap.i2c[3].HOST_FIFO_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:16 | `FMT_THRESH` | RW | 0x0 | The `FMT_THRESH` interrupt remains asserted while the Controller TX FIFO level is less than this setting. |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:0 | `RX_THRESH` | RW | 0x0 | The `RX_THRESH` interrupt remains asserted while the Controller RX FIFO level is greater than this setting. |

### i2c[3].TARGET_FIFO_CONFIG

*Table 2002. i2c_wrap.i2c[3].TARGET_FIFO_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:16 | `ACQ_THRESH` | RW | 0x0 | The `ACQ_THRESH` interrupt remains asserted while the Target RX FIFO level is greater than this setting. |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:0 | `TX_THRESH` | RW | 0x0 | The `TX_THRESH` interrupt remains asserted while the Target TX FIFO level is less than this setting. |

### i2c[3].HOST_FIFO_STATUS

*Table 2003. i2c_wrap.i2c[3].HOST_FIFO_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:16 | `RXLVL` | R | - | Controller RX FIFO fill level. |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:0 | `FMTLVL` | R | - | Controller TX FIFO fill level. |

### i2c[3].TARGET_FIFO_STATUS

*Table 2004. i2c_wrap.i2c[3].TARGET_FIFO_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:16 | `ACQLVL` | R | 0x0 | Target RX FIFO fill level. |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:0 | `TXLVL` | R | 0x0 | Target TX FIFO fill level. |

### i2c[3].OVRD

*Table 2005. i2c_wrap.i2c[3].OVRD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2 | SDAVAL | RW | 0x0 | Global control: SDA override value: * 0 - Pull the SDA line low * 1 - Release the SDA line |
| 1 | SCLVAL | RW | 0x0 | Global control: SCL override value: * 0 - Pull the SCL line low * 1 - Release the SCL line |
| 0 | TXOVRDEN | RW | 0x0 | Global control: enables control of the SDA and SCL lines through the SDA_VAL and SCL_VAL fields, respectively. |

### i2c[3].VAL

*Table 2006. i2c_wrap.i2c[3].VAL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | SDA_RX | R | 0x0 | Global status: contains the last 16 SDA over-sampled values. LSB is most recent. |
| 15:0 | SCL_RX | R | 0x0 | Global status: contains the last 16 SCL over-sampled values. LSB is most recent. |

### i2c[3].TIMING0

*Table 2007. i2c_wrap.i2c[3].TIMING0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | Reserved | - | - | Reserved |
| 28:16 | TLOW | RW | 0x0 | Global control: specifies the LOW period of SCL (t_LOW) in system clock cycles. Must be  2. See Table 11 in the I2C Specification for more details. This field is sized to meet the I2C Standard-mode (100 kHz) minimum t_LOW = 4.7 µs requirement, assuming a 1 GHz system clock. |
| 15:13 | Reserved | - | - | Reserved |
| 12:0 | THIGH | RW | 0x0 | Global control: specifies the HIGH period of SCL (t_HIGH) in system clock cycles. Must be  2. See Table 11 in the I²C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s minimum t_HIGH = 4.0 µs requirement, assuming a 1 GHz system clock. |

### i2c[3].TIMING1

*Table 2008. i2c_wrap.i2c[3].TIMING1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | Reserved | - | - | Reserved |
| 24:16 | T_F | RW | 0x0 | Global control: specifies the fall time of SDA and SCL (t_f) in system clock cycles. The fall time is measured from 70% to 30% of the signal swing. See Table 11 in the I2C Specification for more details. This field is sized to meet I2C Standard-mode (100 kHz)'s maximum t_f = 300 ns requirement, assuming a 1 GHz system clock. |
| 15:10 | Reserved | - | - | Reserved |
| 9:0 | T_R | RW | 0x0 | Global control: specifies the rise time of SDA and SCL (t_r) in system clock cycles. The rise time is measured from 30% to 70% of the signal swing. See Table 11 in the I2C Specification for more details. This field is sized to meet I2C Standard-mode (100 kHz)'s maximum t_r = 1000 ns requirement, assuming a 1 GHz system clock. |

### i2c[3].TIMING2

*Table 2009. i2c_wrap.i2c[3].TIMING2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `THD_STA` | RW | 0x0 | Global control: specifies the setup time for a (repeated) START condition (`t_HD;STA`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s minimum `t_HD;STA = 4.0 µs` requirement, assuming a 1 GHz system clock. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12:0 | `TSU_STA` | RW | 0x0 | Global control: specifies the setup time for a repeated START condition (`t_SU;STA`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s minimum `t_SU;STA = 4.7 µs` requirement, assuming a 1 GHz system clock. |

### i2c[3].TIMING3

*Table 2010. i2c_wrap.i2c[3].TIMING3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `THD_DAT` | RW | 0x0 | Global control: specifies the data and (N)ACK bits hold time (`t_HD;DAT`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s `t_HD;DAT = 5.0 µs` minimum requirement, assuming a 1 GHz system clock. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8:0 | `TSU_DAT` | RW | 0x0 | Global control: specifies the data setup time (`t_SU;DAT`) in system clock cycles. See Table 11 in the I2C Specification for details. This field is sized to meet the I2C Standard-mode (100 kHz)'s `t_SU;DAT = 250 ns` minimum requirement, assuming a 1 GHz system clock. |

### i2c[3].TIMING4

*Table 2011. i2c_wrap.i2c[3].TIMING4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:16 | `T_BUF` | RW | 0x0 | Global control: specifies the time between a STOP and START condition (`t_BUF`) in system clock cycles. See Table 11 in the I²C Specification for details. This field is sized to meet I2C Standard-mode (100kHz)'s `t_BUF = 4.7 µs` minimum requirement, assuming a 1 GHz system clock. |
| 15:13 | `Reserved` | - | - | Reserved |
| 12:0 | `TSU_STO` | RW | 0x0 | Global control: specifies the setup time for a STOP condition (`t_SU;STO`) in system clock cycles. See Table 11 in the I²C Specification for details. This field is sized to meet I2C Standard-mode (100 kHz)'s `t_SU;STO = 4.0 µs` minimum requirement, assuming a 1 GHz system clock. |

### i2c[3].TIMEOUT_CTRL

*Table 2012. i2c_wrap.i2c[3].TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Timeout Enable. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 30 | MODE | RW | 0x0 | Global control: timeout mode. * 0 - Stretch Timeout (Controller Mode only). If the target stretches the clock for more time than TIMEOUT_CTRL.VAL, the STRETCH_TIMEOUT interrupt will be asserted. * 1 - Bus Timeout. If SCL is LOW for more time than TIMEOUT_CTRL.VAL: * In Controller Mode, the CONTROLLER_EVENTS.BUS_TIMEOUT flag will be asserted, triggering the CONTROLLER_HALT interrupt. * In Target Mode, the TARGET_EVENTS.BUS_TIMEOUT flag will be set, triggering the TX_STRETCH interrupt. |
| 29:0 | VAL | RW | 0x0 | Global control: Specifies the timeout value in system clock cycles. The meaning of this field depends on MODE. |

### i2c[3].TARGET_ID

*Table 2013. i2c_wrap.i2c[3].TARGET_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | - | - | Reserved |
| 27:21 | MASK1 | RW | 0x0 | ADDRESS1 mask. |
| 20:14 | ADDRESS1 | RW | 0x0 | Target Address 1. This target responds if the 7-bit address matches ADDRESS1 & MASK1. MASK1 = 0x0 disables this address. |
| 13:7 | MASK0 | RW | 0x0 | ADDRESS0 mask. |
| 6:0 | ADDRESS0 | RW | 0x0 | Target Address 0. This target responds if the 7-bit address matches ADDRESS0 & MASK0. MASK0 = 0x0 disables this address. |

### i2c[3].ACQDATA

*Table 2014. i2c_wrap.i2c[3].ACQDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | Reserved | - | - | Reserved |
| 10:8 | SIGNAL | R | 0x0 | This field indicates if this FIFO entry represents/is associated with control signal(s): * 0x0 - The entry is an ordinary data byte that has been ACKed. * 0x1 - The entry is an address byte preceded by a START. * 0x2 - The entry is a STOP condition following ACKed data bytes. * 0x3 - The entry is an address byte preceded by a repeated START. * 0x4 - The entry is a NACKed data byte. * 0x5 - The entry is an address byte preceded by a (repeated) START. However, the data bytes that followed this address byte were NACKed. * 0x6 - Error. A transaction preceding ended abnormally, for example, due to an unexpected STOP condition, a Bus or Stretch Timeout, a software NACK, or a lost arbitration.<br><br>If the FIFO does not have enough space to record a complete transaction, an Error (0x6) entry may appear alone. |
| 7:0 | ABYTE | R RMOD | 0x0 | Reading this register pops the ACQ FIFO. The meaning of this field depends on SIGNAL: * SIGNAL = 0x0, 0x1, 0x3, 0x4, or 0x5 - This field contains an address or data byte sent by the controller. * SIGNAL = 0x2, 0x6 - This field is meaningless. |

### i2c[3].TXDATA

*Table 2015. i2c_wrap.i2c[3].TXDATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | DATA | W WMOD | 0x0 | Writing to this register pushes data into the Target TX FIFO. The controller reads data from this FIFO during a READ transaction. |

### i2c[3].HOST_TIMEOUT_CTRL

*Table 2016. i2c_wrap.i2c[3].HOST_TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30:0 | `VAL` | RW | 0x0 | Target/Monitor Mode control: controller clock generation timeout value specified in system clock cycles. * Target Mode - If the controller stops generating the clock for more time than this setting, this target asserts the Controller Timeout Interrupt. * Monitor Mode - this field is required to be nonzero for the Bus Monitor to transition out of the initial Busy state. Set this field to `0x0` to disable this behavior. |

### i2c[3].TARGET_TIMEOUT_CTRL

*Table 2017. i2c_wrap.i2c[3].TARGET_TIMEOUT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Enable Target Timeout. |
| 30:0 | `VAL` | RW | 0x0 | When this target has stretched the clock for more time than this setting, this target will NACK incoming data bytes or release the SDA line for outgoing data bytes. The count is cumulative over an entire transaction. In other words, this is SMBus's cumulative target clock extension time.<br><br>The behavior for the address byte is configurable via `CTRL.NACK_ADDR_AFTER_TIMEOUT`. |

### i2c[3].TARGET_NACK_COUNT

*Table 2018. i2c_wrap.i2c[3].TARGET_NACK_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `TARGET_NACK_COUNT` | RW RC | 0x0 | Indicates the number of transactions NACKed by this target since the last read of this register, saturating at 255. This field can be used to track how many transactions were missed when the Target RX FIFO is full. |

### i2c[3].TARGET_ACK_CTRL

*Table 2019. i2c_wrap.i2c[3].TARGET_ACK_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `NACK` | W 1P | 0x0 | When `STATUS.ACK_CTRL_STRETCH = 1`, writing `1` to this field causes this target to NACK all bytes of the transaction. Effective only when `CTRL.ACK_CTRL_EN = 1`. |
| 30:9 | `Reserved` | - | - | Reserved |
| 8:0 | `NBYTES` | RW | 0x0 | When `STATUS.ACK_CTRL_STRETCH = 1`, writing to this register specifies the number of bytes this target should ACK. The count decrements per byte ACKed. Effective only when `CTRL.ACK_CTRL_EN = 1`. See `CTRL.ACK_CTRL_EN` for details.<br><br>This control field is used to implement SMBus's mid-transfer (N)ACK responses. |

### i2c[3].ACQ_FIFO_NEXT_DATA

*Table 2020. i2c_wrap.i2c[3].ACQ_FIFO_NEXT_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R | 0x0 | This field contains the next byte to be pushed into the Target RX FIFO, allowing software to decide whether to accept or reject it. Valid only when `STATUS.ACK_CTRL_STRETCH = 1`. See `CTRL.ACK_CTRL_EN` for details. |

### i2c[3].HOST_NACK_HANDLER_TIMEOUT

*Table 2021. i2c_wrap.i2c[3].HOST_NACK_HANDLER_TIMEOUT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `EN` | RW | 0x0 | Enables Controller NACK Timeout. |
| 30:0 | `VAL` | RW | 0x0 | Timeout value (specified in system clock cycles) for this controller Mode to automatically STOP a transaction when `CONTROLLER_EVENTS.NACK` is set. |

### i2c[3].CONTROLLER_EVENTS

*Table 2022. i2c_wrap.i2c[3].CONTROLLER_EVENTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `ARBITRATION_LOST` | RW1C | 0x0 | Indicates that the controller has halted due to it losing an arbitration against another controller. This bit triggers the `CONTROLLER_HALT` interrupt. Writing `1` clears this bit. |
| 2 | `BUS_TIMEOUT` | RW1C | 0x0 | Indicates that this controller has halted due to a Bus Timeout. See `TIMEOUT_CTRL` for details. This bit triggers the `CONTROLLER_HALT` interrupt. Writing `1` clears this bit. |
| 1 | `UNHANDLED_NACK_TIMEOUT` | RW1C | 0x0 | Indicates that this controller has halted due to a Controller NACK Timeout. See `HOST_NACK_HANDLER_TIMEOUT` for details. This bit triggers the `CONTROLLER_HALT` interrupt. Writing `1` clears this bit. |
| 0 | `NACK` | RW1C | 0x0 | Indicates that this controller has halted due to an unexpected NACK sent by the target. This behavior can be disabled by writing `1` to `FDATA.NAKOK`. This bit triggers the `CONTROLLER_HALT` interrupt. Writing `1` clears this bit. |

### i2c[3].TARGET_EVENTS

*Table 2023. i2c_wrap.i2c[3].TARGET_EVENTS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `STOP_DETECT` | RW1C | 0x0 | Stop Detect Flag (Target Mode only). Set to 1 by hardware when a STOP is detected while `CTRL.ENABLETARGET = 1`. Cleared when software writes 1. |
| 3 | `START_DETECT` | RW1C | 0x0 | Start Detect Flag (Target Mode only). Set to 1 by hardware when a START (or repeated START) is detected while `CTRL.ENABLETARGET = 1`. Cleared when software writes 1. |
| 2 | `ARBITRATION_LOST` | RW1C | 0x0 | Indicates that a controller has lost arbitration, causing a READ transaction to end. This bit triggers the `TX_STRETCH` interrupt. Writing `1` clears this bit. |
| 1 | `BUS_TIMEOUT` | RW1C | 0x0 | Indicates that this target has halted due a Bus Timeout terminating a READ transaction. See `TIMEOUT_CTRL` for details. This bit triggers the `TX_STRETCH` interrupt. Writing `1` clears this bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `TX_PENDING` | RW1C | 0x0 | Indicates that the target is stretching the clock due to receiving a READ address byte and waiting for software to confirm the release of the Target TX FIFO data. Valid only if `CTRL.TX_STRETCH_CTRL_EN = 1`. See `CTRL.TX_STRETCH_CTRL_EN` for details. This bit triggers the `TX_STRETCH` interrupt. Writing `1` clears this bit. |

### i2c[3].SMBUS_STATUS

*Table 2024. i2c_wrap.i2c[3].SMBUS_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `SMBALERT` | R | 0x0 | Controller Mode status: indicates that the `smbalert_ni` input is asserted. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `SMBSUS` | R | 0x0 | Target Mode status: indicates that the `smbsus_ni` input is asserted. |

### I2C_CTRL[3]

*Table 2025. i2c_wrap.I2C_CTRL[3] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `SMBUS_EN` | RW | 0x0 | SMBus Enable. Gates the SMBus Alert signal. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `I2C_CONTROLLER_MODE_EN` | RW | 0x0 | SMBus Pad Direction. When set, the SMBus alert pad is an input and the SMBus suspend pad is an output, as for an SMBus host. When clear, the alert pad is an output and the suspend pad is an input, as for an SMBus target. SCL and SDA are unaffected. Program it to agree with the core's CTRL.ENABLEHOST and CTRL.ENABLETARGET setting. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `I2C_EN` | RW | 0x0 | I2C Pad Function Select. When set, the integration routes this instance's SCL, SDA, SMBus alert and SMBus suspend signals to their pads. Does not enable the I2C core; CTRL.ENABLEHOST and CTRL.ENABLETARGET select the core's operating mode. |

## I3C Controller

The I3C controller is based on the CHIPS Alliance i3c-core. Controller and target behavior, the HCI queues, common command codes, and the bus protocol are specified by that design.

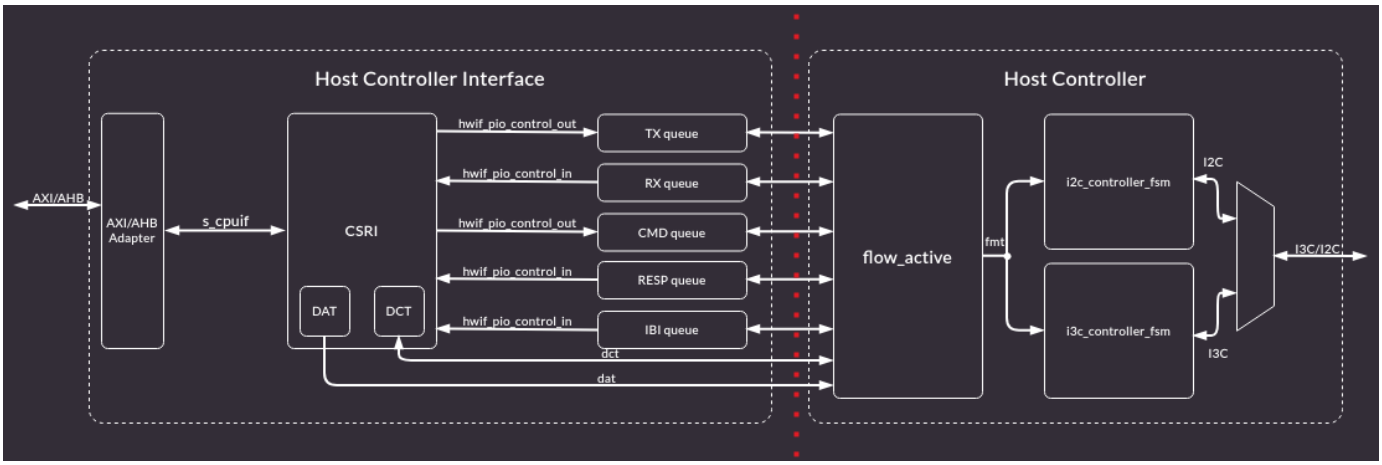


*Figure 70. CHIPS Alliance i3c-core reference architecture*

Figure 70 shows the upstream core.

**OCAH Modifications**

When target mode is used, budget four core-clock cycles for the internal SCL-to-SDA response. A 12 ns turnaround limit requires approximately 333.3 MHz or faster before pad and routing allowance; see the Peripheral Timing Requirements section and the upstream PHY timing requirements. Controller-mode timing examples at lower frequencies do not demonstrate compliance with this target-mode limit.

SMC instantiates six controllers. `i3ccore_wrapper` exposes one AXI4-Lite slave and selects the instance from the address. Instances are spaced by `0x1000`, and each instance is presented with a 12-bit offset inside that window.

`i3c_wrapper` drives the core's AXI4 register port as a single beat: `awlen` and `arlen` are 0 and `wlast` is 1. The core's controller-mode SCL and SDA outputs are bus levels. The wrapper turns those into open-drain pad enables: SCL is driven only low, and SDA is driven in both directions while push-pull mode is selected.

Each instance brings out its device-address, device-characteristic, and reverse-lookup memories, one interrupt, and the recovery pins.

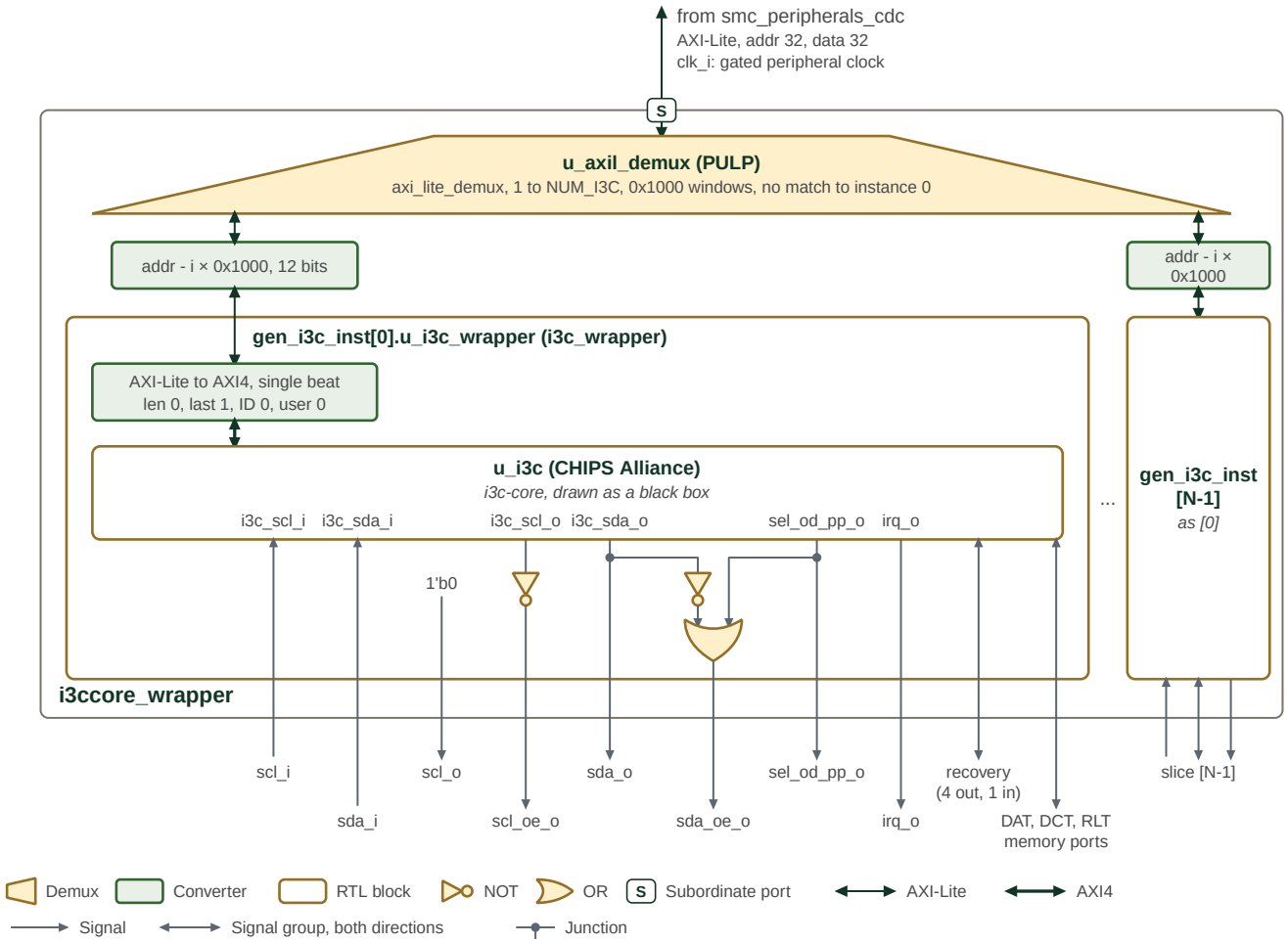


*Figure 71. OCAH I3C wrapper*

Figure 71 shows that wrapper around the CHIPS Alliance core, which it draws as a black box. Parentheses after an instance name give its module or, for a block taken from another project, that project; the demux is a PULP `axi_lite_demux`. `i3ccore_wrapper` runs on one clock, `clk_i`; in the SMC that is the gated peripheral clock, and its AXI4-Lite port, with 32-bit address and data, arrives through `smc_peripherals_cdc`. The demux `u_axil_demux` registers every channel and allows eight outstanding transactions. It has no error slave: an address outside every instance window reaches instance 0. Each generate block subtracts its index times `0x1000` from the address before `i3c_wrapper` sees it.

Inside `i3c_wrapper`, the converter tie-offs are `awlen`/`arlen` 0, `wlast` 1, and zero ID and user. The gates below the core are the pad-enable logic: `scl_o` is tied low and `scl_oe_o` is the inverse of the core's SCL level, so the pad only pulls SCL low; `sda_o` is the core's SDA level and `sda_oe_o` is `sel_od_pp_o` OR the inverse of `sda_o`. The core's own SDA output enable, which it drives only in target mode, is not used.

`irq_o`, the five recovery signals and the DAT, DCT and reverse-lookup memory ports pass straight through to per-instance vectors on the `i3ccore_wrapper` boundary. In the SMC `recovery_payload_available_o`, `recovery_image_activated_o`, `peripheral_reset_o` and `escalated_reset_o` are left unconnected, `peripheral_reset_done_i` is tied low, and the interrupts reach peripheral interrupts 12 to 17.

The register map below reproduces the upstream SystemRDL descriptions; those under `I3C_EC` are upstream's own rather than MIPI I3C HCI text. The fields below differ from or go beyond those descriptions.

These read back the value written but reach no logic:

- `PIOControl.PIO_CONTROL.ENABLE`: the PIO queues are always enabled.

- `I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL.TARGET_XACT_ENABLE`: target transactions cannot be disabled.
- `I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL.RSTACT_DEFBYTE_02`: RSTACT with defining byte 0x02 is always ACKed.
- `I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS.RESET_TIME_PERIPHERAL` and `RESET_TIME_TARGET`: GET RSTACT with defining byte 0x81 or 0x82 returns 0xFF.
- `I3C_EC.SoCMgmtIf.REC_INTF_CFG.REC_INTF_BYPASS`: `DISABLE_LOOPBACK` ties the bypass off, so recovery always goes through the I3C core.

These misreport the core:

- `I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES.DAA_ENTDAA_SUPPORT` reads 0, but ENTDAA is supported.
- `I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL.STBY_CR_ENABLE_INIT` mode `2'b11` behaves as `2'b01` (active controller), not Hot-Join-then-standby.

These behave as follows, which the descriptions leave unstated:

- `I3C_EC.TTI.TARGET_ERR_CNT_TE3` counts parity errors on the ENTDAA-assigned dynamic address; `TARGET_ERR_CNT_TE4` counts reserved bytes other than `7'h7E` with R/W=1.
- `I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW` and `TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW`: the enable in `TARGET_ERR_CTRL` gates detection, so an overflow while it is clear is neither counted nor interrupts.
- `PIOControl.DATA_BUFFER_THLD_CTRL`: `RX_START_THLD` and `TX_BUF_THLD` test free space, `TX_START_THLD` and `RX_BUF_THLD` stored entries; each encodes `2^(N+1)` entries.
- `PIOControl.QUEUE_THLD_CTRL`: raw entry counts, free for `CMD_EMPTY_BUF_THLD` and stored for `RESP_BUF_THLD` and `IBI_STATUS_THLD`. Only these ready thresholds are implemented; start thresholds are unconnected.
- `DCT.DCT_MEMORY`: `PID_LO` holds the first two (most significant) ENTDAA PID bytes and `PID_HI` the next four. `DYNAMIC_ADDRESS` holds the address in bits 7:1 and its odd parity in bit 0.

Consult the i3c-core documentation for the controller. The register map of this wrapper follows.

#### Memory Map and Register Reference

#### Memory Map

#### Integration Limits

The SoC Management Interface header at `0x300` reports `CAP_LENGTH = 0x1A` DWORDs. This is inconsistent with the next capability header at `0x398`: the span is `0x98` bytes, or `0x26` DWORDs. A generic walker that trusts the reported length advances to `0x368`, inside the timing registers, and misinterprets data as a capability header. Software for this implementation must use the offsets in this paragraph rather than discovering that boundary from `CAP_LENGTH`.

The DAT and DCT capability fields advertise `TABLE_SIZE = 127`, but only 16 independent entries are implemented. Index decoding retains four bits, so higher window slots alias lower entries rather than providing extra storage. For example, DAT slot 16 at `0x480` aliases slot 0 at `0x400`; DCT slot 16 at `0x900` aliases slot 0 at `0x800`. Limit accesses to indexes 0–15. The complete windows in the generated map describe address decoding, not independent 128-entry capacity.

#### Detailed Register Map

#### OCA I3C Controller Wrapper Address Map

Wrapper address map instantiating the per-controller I3C CSR blocks.

NOTE

**Register Arrays:**

- **i3c_csr[6].I3CBase.HCI_VERSION**: 6 registers, base 0x0, stride 0x1000
- **i3c_csr[6].I3CBase.HC_CONTROL**: 6 registers, base 0x4, stride 0x1000
- **i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR**: 6 registers, base 0x8, stride 0x1000
- **i3c_csr[6].I3CBase.HC_CAPABILITIES**: 6 registers, base 0xC, stride 0x1000
- **i3c_csr[6].I3CBase.RESET_CONTROL**: 6 registers, base 0x10, stride 0x1000
- **i3c_csr[6].I3CBase.PRESENT_STATE**: 6 registers, base 0x14, stride 0x1000
- **i3c_csr[6].I3CBase.INTR_STATUS**: 6 registers, base 0x20, stride 0x1000
- **i3c_csr[6].I3CBase.INTR_STATUS_ENABLE**: 6 registers, base 0x24, stride 0x1000
- **i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE**: 6 registers, base 0x28, stride 0x1000
- **i3c_csr[6].I3CBase.INTR_FORCE**: 6 registers, base 0x2C, stride 0x1000
- **i3c_csr[6].I3CBase.DAT_SECTION_OFFSET**: 6 registers, base 0x30, stride 0x1000
- **i3c_csr[6].I3CBase.DCT_SECTION_OFFSET**: 6 registers, base 0x34, stride 0x1000
- **i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET**: 6 registers, base 0x38, stride 0x1000
- **i3c_csr[6].I3CBase.PIO_SECTION_OFFSET**: 6 registers, base 0x3C, stride 0x1000
- **i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET**: 6 registers, base 0x40, stride 0x1000
- **i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN**: 6 registers, base 0x4C, stride 0x1000
- **i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL**: 6 registers, base 0x58, stride 0x1000
- **i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL**: 6 registers, base 0x5C, stride 0x1000
- **i3c_csr[6].I3CBase.DEV_CTX_BASE_LO**: 6 registers, base 0x60, stride 0x1000
- **i3c_csr[6].I3CBase.DEV_CTX_BASE_HI**: 6 registers, base 0x64, stride 0x1000
- **i3c_csr[6].I3CBase.DEV_CTX_SG**: 6 registers, base 0x68, stride 0x1000
- **i3c_csr[6].PIOControl.COMMAND_PORT**: 6 registers, base 0x80, stride 0x1000
- **i3c_csr[6].PIOControl.RESPONSE_PORT**: 6 registers, base 0x84, stride 0x1000
- **i3c_csr[6].PIOControl.TX_DATA_PORT**: 6 registers, base 0x88, stride 0x1000
- **i3c_csr[6].PIOControl.RX_DATA_PORT**: 6 registers, base 0x88, stride 0x1000
- **i3c_csr[6].PIOControl.IBI_PORT**: 6 registers, base 0x8C, stride 0x1000
- **i3c_csr[6].PIOControl.QUEUE_THLD_CTRL**: 6 registers, base 0x90, stride 0x1000
- **i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL**: 6 registers, base 0x94, stride 0x1000
- **i3c_csr[6].PIOControl.QUEUE_SIZE**: 6 registers, base 0x98, stride 0x1000
- **i3c_csr[6].PIOControl.ALT_QUEUE_SIZE**: 6 registers, base 0x9C, stride 0x1000
- **i3c_csr[6].PIOControl.PIO_INTR_STATUS**: 6 registers, base 0xA0, stride 0x1000
- **i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE**: 6 registers, base 0xA4, stride 0x1000
- **i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE**: 6 registers, base 0xA8, stride 0x1000
- **i3c_csr[6].PIOControl.PIO_INTR_FORCE**: 6 registers, base 0xAC, stride 0x1000
- **i3c_csr[6].PIOControl.PIO_CONTROL**: 6 registers, base 0xB0, stride 0x1000

- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER**: 6 registers, base 0x100, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0**: 6 registers, base 0x104, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1**: 6 registers, base 0x108, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2**: 6 registers, base 0x10C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3**: 6 registers, base 0x110, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0**: 6 registers, base 0x114, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1**: 6 registers, base 0x118, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2**: 6 registers, base 0x11C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3**: 6 registers, base 0x120, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4**: 6 registers, base 0x124, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5**: 6 registers, base 0x128, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED**: 6 registers, base 0x12C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0**: 6 registers, base 0x130, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1**: 6 registers, base 0x134, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET**: 6 registers, base 0x138, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL**: 6 registers, base 0x13C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS**: 6 registers, base 0x140, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS**: 6 registers, base 0x144, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0**: 6 registers, base 0x148, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1**: 6 registers, base 0x14C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0**: 6 registers, base 0x150, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1**: 6 registers, base 0x154, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2**: 6 registers, base 0x158, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3**: 6 registers, base 0x15C, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4**: 6 registers, base 0x160, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED**: 6 registers, base 0x164, stride 0x1000
- **i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA**: 6 registers, base 0x168, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER**: 6 registers, base 0x180, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL**: 6 registers, base 0x184, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR**: 6 registers, base 0x188, stride 0x1000

- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES**: 6 registers, base 0x18C, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR**: 6 registers, base 0x190, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS**: 6 registers, base 0x194, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR**: 6 registers, base 0x198, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO**: 6 registers, base 0x19C, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS**: 6 registers, base 0x1A0, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO**: 6 registers, base 0x1A4, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE**: 6 registers, base 0x1A8, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE**: 6 registers, base 0x1AC, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS**: 6 registers, base 0x1B0, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS**: 6 registers, base 0x1B4, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR**: 6 registers, base 0x1B8, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL**: 6 registers, base 0x1BC, stride 0x1000
- **i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL**: 6 registers, base 0x1C0, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER**: 6 registers, base 0x200, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.CONTROL**: 6 registers, base 0x204, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.STATUS**: 6 registers, base 0x208, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.RESET_CONTROL**: 6 registers, base 0x20C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS**: 6 registers, base 0x210, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH**: 6 registers, base 0x214, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH**: 6 registers, base 0x218, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH**: 6 registers, base 0x21C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS**: 6 registers, base 0x220, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE**: 6 registers, base 0x224, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE**: 6 registers, base 0x228, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL**: 6 registers, base 0x22C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS**: 6 registers, base 0x230, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE**: 6 registers, base 0x234, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE**: 6 registers, base 0x238, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0**: 6 registers, base 0x23C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1**: 6 registers, base 0x240, stride 0x1000

- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2**: 6 registers, base 0x244, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3**: 6 registers, base 0x248, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4**: 6 registers, base 0x24C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5**: 6 registers, base 0x250, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING**: 6 registers, base 0x254, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC**: 6 registers, base 0x258, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH**: 6 registers, base 0x25C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY**: 6 registers, base 0x260, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED**: 6 registers, base 0x264, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW**: 6 registers, base 0x268, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW**: 6 registers, base 0x26C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT**: 6 registers, base 0x270, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT**: 6 registers, base 0x274, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT**: 6 registers, base 0x278, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT**: 6 registers, base 0x27C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.IBI_PORT**: 6 registers, base 0x280, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE**: 6 registers, base 0x284, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE**: 6 registers, base 0x288, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL**: 6 registers, base 0x28C, stride 0x1000
- **i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL**: 6 registers, base 0x290, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER**: 6 registers, base 0x300, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL**: 6 registers, base 0x304, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS**: 6 registers, base 0x308, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG**: 6 registers, base 0x30C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS**: 6 registers, base 0x310, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2**: 6 registers, base 0x314, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3**: 6 registers, base 0x318, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF**: 6 registers, base 0x31C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR**: 6 registers, base 0x320, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2**: 6 registers, base 0x324, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3**: 6 registers, base 0x328, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG**: 6 registers, base 0x32C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG**: 6 registers, base 0x330, stride 0x1000

- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG**: 6 registers, base 0x334, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG**: 6 registers, base 0x338, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG**: 6 registers, base 0x33C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG**: 6 registers, base 0x340, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG**: 6 registers, base 0x344, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG**: 6 registers, base 0x348, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG**: 6 registers, base 0x34C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG**: 6 registers, base 0x350, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG**: 6 registers, base 0x354, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG**: 6 registers, base 0x358, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG**: 6 registers, base 0x35C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG**: 6 registers, base 0x360, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG**: 6 registers, base 0x364, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG**: 6 registers, base 0x368, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG**: 6 registers, base 0x36C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG**: 6 registers, base 0x370, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG**: 6 registers, base 0x374, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG**: 6 registers, base 0x378, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG**: 6 registers, base 0x37C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG**: 6 registers, base 0x380, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG**: 6 registers, base 0x384, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG**: 6 registers, base 0x388, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG**: 6 registers, base 0x38C, stride 0x1000
- **i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG**: 6 registers, base 0x390, stride 0x1000
- **i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER**: 6 registers, base 0x398, stride 0x1000
- **i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG**: 6 registers, base 0x39C, stride 0x1000
- **i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER**: 6 registers, base 0x3A0, stride 0x1000
- **i3c_csr[6].DAT.DAT_MEMORY[128]**: 768 registers, base 0x400 (i3c_csr x6 stride 0x1000, DAT_MEMORY x128 stride 0x8)
- **i3c_csr[6].DCT.DCT_MEMORY[128]**: 768 registers, base 0x800 (i3c_csr x6 stride 0x1000, DCT_MEMORY x128 stride 0x10)

*Table 2026. OCA I3C Controller Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x5000 | i3c_csr[6].I3CBase.HCI_VERSION | R | Identifies the implemented MIPI I3C HCI version. |
| 0x4 - 0x5004 | i3c_csr[6].I3CBase.HC_CONTROL | RW | Controls host-controller bus operation, mode selection, transaction abort, and Hot-Join handling. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x8 - 0x5008 | i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR | RW | Configures the controller dynamic address and indicates whether that address is valid. |
| 0xC - 0x500C | i3c_csr[6].I3CBase.HC_CAPABILITIES | R | Reports the host-controller capabilities implemented by this instance. |
| 0x10 - 0x5010 | i3c_csr[6].I3CBase.RESET_CONTROL | RW | Requests a controller soft reset or selective PIO queue and FIFO resets. Reset requests are self-clearing. |
| 0x14 - 0x5014 | i3c_csr[6].I3CBase.PRESENT_STATE | R | Reports current controller state, including active-controller bus ownership. |
| 0x20 - 0x5020 | i3c_csr[6].I3CBase.INTR_STATUS | RW | Reports host-controller error and exceptional-condition status. |
| 0x24 - 0x5024 | i3c_csr[6].I3CBase.INTR_STATUS_ENABLE | RW | Enables individual host-controller interrupt status sources. |
| 0x28 - 0x5028 | i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE | RW | Routes enabled host-controller interrupt sources to the interrupt output. |
| 0x2C - 0x502C | i3c_csr[6].I3CBase.INTR_FORCE | W | Forces host-controller interrupts for software testing. |
| 0x30 - 0x5030 | i3c_csr[6].I3CBase.DAT_SECTION_OFFSET | R | Reports the Device Address Table section offset and size. |
| 0x34 - 0x5034 | i3c_csr[6].I3CBase.DCT_SECTION_OFFSET | RW | Configures the Device Characteristic Table section offset, size, and ENTDAA index. |
| 0x38 - 0x5038 | i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET | R | - |
| 0x3C - 0x503C | i3c_csr[6].I3CBase.PIO_SECTION_OFFSET | R | Reports the PIO section offset. |
| 0x40 - 0x5040 | i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET | R | - |
| 0x4C - 0x504C | i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN | R | - |
| 0x58 - 0x5058 | i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL | RW | - |
| 0x5C - 0x505C | i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL | RW | - |
| 0x60 - 0x5060 | i3c_csr[6].I3CBase.DEV_CTX_BASE_LO | RW | - |
| 0x64 - 0x5064 | i3c_csr[6].I3CBase.DEV_CTX_BASE_HI | RW | - |
| 0x68 - 0x5068 | i3c_csr[6].I3CBase.DEV_CTX_SG | R | - |
| 0x80 - 0x5080 | i3c_csr[6].PIOControl.COMMAND_PORT | W | Accepts a 64-bit command descriptor as two consecutive 32-bit writes. The second write enqueues the descriptor. |
| 0x84 - 0x5084 | i3c_csr[6].PIOControl.RESPONSE_PORT | R | Returns one 32-bit response descriptor from the response queue. |
| 0x88 - 0x5088 | i3c_csr[6].PIOControl.TX_DATA_PORT | W | Writes transmit data to the TX FIFO. This port shares its address with the read-only RX data port. |
| 0x88 - 0x5088 | i3c_csr[6].PIOControl.RX_DATA_PORT | R | Reads received data from the RX FIFO. This port shares its address with the write-only TX data port. |
| 0x8C - 0x508C | i3c_csr[6].PIOControl.IBI_PORT | R | Returns IBI status and payload data. Payload data is padded to a 4-byte boundary. |
| 0x90 - 0x5090 | i3c_csr[6].PIOControl.QUEUE_THLD_CTRL | RW | Configures command, response, and IBI queue thresholds as raw entry counts. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x94 - 0x5094 | i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL | RW | Configures TX and RX start and ready thresholds, each encoding 2^(N+1) entries. |
| 0x98 - 0x5098 | i3c_csr[6].PIOControl.QUEUE_SIZE | R | Reports command, response, IBI, TX, and RX queue depths. |
| 0x9C - 0x509C | i3c_csr[6].PIOControl.ALT_QUEUE_SIZE | R | Reports extended IBI and alternate response queue sizes. |
| 0xA0 - 0x50A0 | i3c_csr[6].PIOControl.PIO_INTR_STATUS | RW | Reports PIO queue-threshold and transfer-event interrupt status. |
| 0xA4 - 0x50A4 | i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE | RW | Enables individual PIO interrupt status sources. |
| 0xA8 - 0x50A8 | i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE | RW | Routes enabled PIO interrupt sources to the interrupt output. |
| 0xAC - 0x50AC | i3c_csr[6].PIOControl.PIO_INTR_FORCE | W | - |
| 0xB0 - 0x50B0 | i3c_csr[6].PIOControl.PIO_CONTROL | RW | Controls run/stop and forced abort of enqueued commands. |
| 0x100 - 0x5100 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER | R | - |
| 0x104 - 0x5104 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0 | R | - |
| 0x108 - 0x5108 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1 | R | - |
| 0x10C - 0x510C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2 | RW | - |
| 0x110 - 0x5110 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3 | RW | - |
| 0x114 - 0x5114 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0 | RW | - |
| 0x118 - 0x5118 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1 | RW | - |
| 0x11C - 0x511C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2 | RW | - |
| 0x120 - 0x5120 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3 | RW | - |
| 0x124 - 0x5124 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4 | RW | - |
| 0x128 - 0x5128 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5 | RW | - |
| 0x12C - 0x512C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED | R | - |
| 0x130 - 0x5130 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0 | RW | - |
| 0x134 - 0x5134 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1 | RW | - |
| 0x138 - 0x5138 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET | RW | For devices which support reset, this register will reset the device or management entity |
| 0x13C - 0x513C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL | RW | - |
| 0x140 - 0x5140 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x144 - 0x5144 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS | RW | - |
| 0x148 - 0x5148 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0 | RW | - |
| 0x14C - 0x514C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1 | RW | - |
| 0x150 - 0x5150 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0 | R | - |
| 0x154 - 0x5154 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1 | R | - |
| 0x158 - 0x5158 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2 | R | - |
| 0x15C - 0x515C | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3 | R | - |
| 0x160 - 0x5160 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4 | R | - |
| 0x164 - 0x5164 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED | R | - |
| 0x168 - 0x5168 | i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA | R | Indirect memory access to address space configured in INDIRECT_FIFO_CTRL at the Head Pointer offset. |
| 0x180 - 0x5180 | i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER | R | - |
| 0x184 - 0x5184 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL | RW | Configures standby-controller operation, dynamic-address-assignment methods, target transactions, and controller-role handoff. |
| 0x188 - 0x5188 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR | RW | Configures standby-controller dynamic and static addresses. |
| 0x18C - 0x518C | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES | R | Reports standby-controller capabilities. |
| 0x190 - 0x5190 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR | RW | - |
| 0x194 - 0x5194 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS | RW | - |
| 0x198 - 0x5198 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR | RW | - |
| 0x19C - 0x519C | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO | RW | - |
| 0x1A0 - 0x51A0 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS | RW | - |
| 0x1A4 - 0x51A4 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO | RW | - |
| 0x1A8 - 0x51A8 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE | RW | When set to 1'b1, and the corresponding interrupt status field is set in register STBY_CR_INTR_STATUS, the Host Controller shall assert an interrupt to the Host. |
| 0x1AC - 0x51AC | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE | RW | For software testing, when set to 1'b1, forces the corresponding interrupt to be sent to the Host, if the corresponding fields are set in register STBY_CR_INTR_SIGNAL_ENABLE |
| 0x1B0 - 0x51B0 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS | RW | - |
| 0x1B4 - 0x51B4 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS | RW | Reports the defining byte of the last RSTACT CCC received and configures the reset recovery times. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x1B8 - 0x51B8 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR | RW | - |
| 0x1BC - 0x51BC | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL | R | - |
| 0x1C0 - 0x51C0 | i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL | R | - |
| 0x200 - 0x5200 | i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER | R | - |
| 0x204 - 0x5204 | i3c_csr[6].I3C_EC.TTI.CONTROL | RW | Control Register |
| 0x208 - 0x5208 | i3c_csr[6].I3C_EC.TTI.STATUS | R | Status Register |
| 0x20C - 0x520C | i3c_csr[6].I3C_EC.TTI.RESET_CONTROL | RW | Reset Control for Queues and IBI Retry Counter |
| 0x210 - 0x5210 | i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS | R | Dynamic queue full/empty status for debug and monitoring |
| 0x214 - 0x5214 | i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH | R | Current number of entries in descriptor queues |
| 0x218 - 0x5218 | i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH | R | Current number of DWORD entries in data queues |
| 0x21C - 0x521C | i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH | R | Current number of entries in IBI queue |
| 0x220 - 0x5220 | i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS | RW | Interrupt Status.<br><br>How a bit is cleared depends on the bit:<br><br>• Auto-cleared by hardware on a specific access: RX_DESC_STAT, RX_DESC_THLD_STAT and RX_DATA_THLD_STAT clear when software reads the corresponding RX queue; IBI_DONE clears when software reads STATUS.LAST_IBI_STATUS. These bits are also write-1-to-clear, but the normal servicing access clears them without an explicit write here.<br>• Write-1-to-clear only (no hardware auto-clear): TX_DESC_COMPLETE, TX_DESC_STAT, TX_DESC_THLD_STAT, TX_DATA_THLD_STAT, IBI_THLD_STAT, and the not-implemented TRANSFER_ERR_STAT, TRANSFER_ABORT_STAT, TX_DESC_TIMEOUT, RX_DESC_TIMEOUT. Write 1 to the bit in this register to clear it.<br>• Read-only, hardware-driven level status that cannot be cleared by software: PENDING_IBI and PENDING_INTERRUPT track live conditions. |
| 0x224 - 0x5224 | i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE | RW | Interrupt Enable |
| 0x228 - 0x5228 | i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE | RW | Interrupt Force |
| 0x22C - 0x522C | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL | RW | Target Error Detection Control. Set bits to enable error detection. Clear to disable. |
| 0x230 - 0x5230 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS | RW | Target Error Interrupt Status. Write 1 to clear individual bits. |
| 0x234 - 0x5234 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE | RW | Target Error Interrupt Enable. Set bits to enable corresponding interrupts. |
| 0x238 - 0x5238 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE | RW | Target Error Interrupt Force. Set bits to force corresponding interrupts for testing. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x23C - 0x523C | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0 | RW | Counts TE0 errors. Saturates at 255. Write 0 to clear. |
| 0x240 - 0x5240 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1 | RW | Counts TE1 errors. Saturates at 255. Write 0 to clear. |
| 0x244 - 0x5244 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2 | RW | Counts TE2 errors. Saturates at 255. Write 0 to clear. |
| 0x248 - 0x5248 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3 | RW | Counts TE3 errors. Saturates at 255. Write 0 to clear. |
| 0x24C - 0x524C | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4 | RW | Counts TE4 errors. Saturates at 255. Write 0 to clear. |
| 0x250 - 0x5250 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5 | RW | Counts TE5 errors. Saturates at 255. Write 0 to clear. |
| 0x254 - 0x5254 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING | RW | Counts framing errors. Saturates at 255. Write 0 to clear. |
| 0x258 - 0x5258 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC | RW | Counts Recovery Interface PEC errors. Saturates at 255. Write 0 to clear. |
| 0x25C - 0x525C | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH | RW | Counts Recovery Interface length mismatch errors. Saturates at 255. Write 0 to clear. |
| 0x260 - 0x5260 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY | RW | Counts Recovery Interface write-to-read-only errors. Saturates at 255. Write 0 to clear. |
| 0x264 - 0x5264 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED | RW | Counts Recovery Interface unsupported command errors. Saturates at 255. Write 0 to clear. |
| 0x268 - 0x5268 | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW | RW | Counts Recovery Interface RX FIFO overflow errors. Saturates at 255. Write 0 to clear. |
| 0x26C - 0x526C | i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW | RW | Counts Recovery Interface INDIRECT FIFO overflow errors. Saturates at 255. Write 0 to clear. |
| 0x270 - 0x5270 | i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT | R | RX Descriptor Queue Port |
| 0x274 - 0x5274 | i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT | R | RX Data Port |
| 0x278 - 0x5278 | i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT | W | TX Descriptor Queue Port |
| 0x27C - 0x527C | i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT | W | TX Data Port |
| 0x280 - 0x5280 | i3c_csr[6].I3C_EC.TTI.IBI_PORT | W | IBI Data Port |
| 0x284 - 0x5284 | i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE | R | Queue Size |
| 0x288 - 0x5288 | i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE | R | IBI Queue Size |
| 0x28C - 0x528C | i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL | RW | Queue Threshold Control |
| 0x290 - 0x5290 | i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL | RW | IBI Queue Threshold Control |
| 0x300 - 0x5300 | i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER | R | - |
| 0x304 - 0x5304 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL | RW | - |
| 0x308 - 0x5308 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x30C - 0x530C | i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG | RW | Configures the recovery interface and signals payload completion to the recovery handler. |
| 0x310 - 0x5310 | i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS | RW | - |
| 0x314 - 0x5314 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2 | RW | - |
| 0x318 - 0x5318 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3 | RW | - |
| 0x31C - 0x531C | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF | RW | - |
| 0x320 - 0x5320 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR | RW | - |
| 0x324 - 0x5324 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2 | RW | - |
| 0x328 - 0x5328 | i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3 | RW | - |
| 0x32C - 0x532C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG | RW | - |
| 0x330 - 0x5330 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG | RW | - |
| 0x334 - 0x5334 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG | RW | - |
| 0x338 - 0x5338 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG | RW | - |
| 0x33C - 0x533C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG | RW | - |
| 0x340 - 0x5340 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG | RW | - |
| 0x344 - 0x5344 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG | RW | - |
| 0x348 - 0x5348 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG | RW | - |
| 0x34C - 0x534C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG | RW | - |
| 0x350 - 0x5350 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG | RW | - |
| 0x354 - 0x5354 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG | RW | - |
| 0x358 - 0x5358 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG | RW | - |
| 0x35C - 0x535C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG | RW | - |
| 0x360 - 0x5360 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG | RW | - |
| 0x364 - 0x5364 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG | RW | - |
| 0x368 - 0x5368 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG | RW | - |
| 0x36C - 0x536C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG | RW | - |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x370 - 0x5370 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG | RW | - |
| 0x374 - 0x5374 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG | RW | - |
| 0x378 - 0x5378 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG | RW | - |
| 0x37C - 0x537C | i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG | RW | - |
| 0x380 - 0x5380 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG | RW | - |
| 0x384 - 0x5384 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG | RW | - |
| 0x388 - 0x5388 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG | RW | - |
| 0x38C - 0x538C | i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG | RW | - |
| 0x390 - 0x5390 | i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG | RW | - |
| 0x398 - 0x5398 | i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER | R | - |
| 0x39C - 0x539C | i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG | R | - |
| 0x3A0 - 0x53A0 | i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER | R | Register after the last EC must advertise ID == 0. Termination register is added to guarantee that the discovery mechanism reaches termination value. |
| 0x400 - 0x57F8 | i3c_csr[6].DAT.DAT_MEMORY[128] | RW | - |
| 0x800 - 0x5FF0 | i3c_csr[6].DCT.DCT_MEMORY[128] | R | Holds one Device Characteristic Table entry per device enumerated by ENTDAA. |

### i3c_csr[6].I3CBase.HCI_VERSION

*Table 2027. oca_i3c_wrap.i3c_csr[6].I3CBase.HCI_VERSION fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `VERSION` | R | 0x120 | - |

### i3c_csr[6].I3CBase.HC_CONTROL

*Table 2028. oca_i3c_wrap.i3c_csr[6].I3CBase.HC_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `BUS_ENABLE` | RW | 0x0 | Host Controller Bus Enable |
| 30 | `RESUME` | RW1C | 0x0 | Host Controller Resume:<br><br>0 - Controller is running<br><br>1 - Controller is suspended<br><br>Write 1 to resume Controller operations. |
| 29 | `ABORT` | RW | 0x0 | Host Controller Abort when set to 1 |
| 28:13 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 12 | HALT_ON_CMD_SEQ_TIMEOUT | RW | 0x0 | Halt on Command Sequence Timeout when set to 1 |
| 11:9 | Reserved | - | - | Reserved |
| 8 | HOT_JOIN_CTRL | RW | 0x0 | Hot-Join ACK/NACK Control:<br><br>0 - ACK Hot-Join request<br><br>1 - NACK Hot-Join request and send Broadcast CCC to disable Hot-Join |
| 7 | I2C_DEV_PRESENT | RW | 0x0 | I2C Device Present on Bus:<br><br>0 - pure I3C bus<br><br>1 - legacy I2C devices on the bus |
| 6 | MODE_SELECTOR | R | 0x1 | DMA/PIO Mode Selector:<br><br>0 - DMA<br><br>1 - PIO |
| 5 | Reserved | - | - | Reserved |
| 4 | DATA_BYTE_ORDER_MODE | R | 0x0 | Data Byte Ordering Mode:<br><br>0 - Little Endian<br><br>1 - Big Endian |
| 3 | AUTOCMD_DATA_RPT | R | 0x0 | Auto-Command Data Report:<br><br>0 - coalesced reporting<br><br>1 - separated reporting |
| 2:1 | Reserved | - | - | Reserved |
| 0 | IBA_INCLUDE | RW | 0x0 | Include I3C Broadcast Address:<br><br>0 - skips I3C Broadcast Address for private transfers<br><br>1 - includes I3C Broadcast Address for private transfers |

### i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR

*Table 2029. oca_i3c_wrap.i3c_csr[6].I3CBase.CONTROLLER_DEVICE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | DYNAMIC_ADDR_VALID | RW | 0x0 | Dynamic Address is Valid:<br><br>0 - dynamic address is invalid<br><br>1 - dynamic address is valid |
| 30:23 | Reserved | - | - | Reserved |
| 22:16 | DYNAMIC_ADDR | RW | 0x0 | Device Dynamic Address |
| 15:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.HC_CAPABILITIES

*Table 2030. oca_i3c_wrap.i3c_csr[6].I3CBase.HC_CAPABILITIES fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30 | `SG_CAPABILITY_DC_EN` | R | 0x0 | Device context memory:<br><br>0 - must be physically continuous<br><br>1 - controller supports scatter-gather |
| 29 | `SG_CAPABILITY_IBI_EN` | R | 0x0 | DMA only: IBI status and IBI Data rings memory:<br><br>0 - must be physically continuous<br><br>1 - controller supports scatter-gather |
| 28 | `SG_CAPABILITY_CR_EN` | R | 0x0 | DMA only: Command and Response rings memory:<br><br>0 - must be physically continuous<br><br>1 - controller supports scatter-gather |
| 27:22 | `Reserved` | - | - | Reserved |
| 21:20 | `CMD_SIZE` | R | 0x0 | Size and structure of the Command Descriptor:<br><br>2’b0: 2 DWORDs,<br><br>all other reserved. |
| 19:14 | `Reserved` | - | - | Reserved |
| 13 | `SCHEDULED_COMMANDS_EN` | R | 0x0 | Controller command scheduling:<br><br>0 - not supported<br><br>1 - supported |
| 12 | `IBI_CREDIT_COUNT_EN` | R | 0x0 | Controller IBI credit count:<br><br>0 - not supported<br><br>1 - supported |
| 11 | `IBI_DATA_ABORT_EN` | R | 0x0 | Controller IBI data abort:<br><br>0 - not supported<br><br>1 - supported |
| 10 | `CMD_CCC_DEFBYTE` | R | 0x1 | CCC with defining byte:<br><br>0 - not supported<br><br>1 - supported |
| 9:8 | `Reserved` | - | - | Reserved |
| 7 | `HDR_TS_EN` | R | 0x0 | HDR-Ternary transfers:<br><br>0 - not supported<br><br>1 - supported |
| 6 | `HDR_DDR_EN` | R | 0x0 | HDR-DDR transfers:<br><br>0 - not supported<br><br>1 - supported |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 5 | STANDBY_CR_CAP | R | 0x0 | Switching from active to standby mode:<br><br>0 - not supported, this controller is always active on I3C<br><br>1- supported, this controller can hand off I3C to secondary controller |
| 4 | Reserved | - | - | Reserved |
| 3 | AUTO_COMMAND | R | 0x0 | Automatic read command on IBI:<br><br>0 - not supported<br><br>1 - supported |
| 2 | COMBO_COMMAND | R | 0x0 | Controller combined command:<br><br>0 - not supported<br><br>1 - supported |
| 1:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.RESET_CONTROL

*Table 2031. oca_i3c_wrap.i3c_csr[6].I3CBase.RESET_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5 | IBI_QUEUE_RST | RW | 0x0 | Clear IBI queue from software. Valid only in PIO mode. |
| 4 | RX_FIFO_RST | RW | 0x0 | Clear RX FIFO from software. Valid only in PIO mode. |
| 3 | TX_FIFO_RST | RW | 0x0 | Clear TX FIFO from software. Valid only in PIO mode. |
| 2 | RESP_QUEUE_RST | RW | 0x0 | Clear response queue from software. Valid only in PIO mode. |
| 1 | CMD_QUEUE_RST | RW | 0x0 | Clear command queue from software. Valid only in PIO mode. |
| 0 | SOFT_RST | RW | 0x0 | Reset controller from software. Not supported currently |

### i3c_csr[6].I3CBase.PRESENT_STATE

*Table 2032. oca_i3c_wrap.i3c_csr[6].I3CBase.PRESENT_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | Reserved | - | - | Reserved |
| 2 | AC_CURRENT_OWN | R | 0x0 | Controller I3C state:<br><br>0 - not bus owner<br><br>1 - bus owner |
| 1:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.INTR_STATUS

*Table 2033. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | Reserved | - | - | Reserved |
| 14 | SCHED_CMD_MISSED_TICK_STAT | RW1C | 0x0 | Scheduled commands could be executed due to controller being busy. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 13 | HC_ERR_CMD_SEQ_TIMEOUT_STAT | RW1C | 0x0 | Command timeout after prolonged stall. |
| 12 | HC_WARN_CMD_SEQ_STALL_STAT | RW1C | 0x0 | Clock stalled due to lack of commands. |
| 11 | HC_SEQ_CANCEL_STAT | RW1C | 0x0 | Controller had to cancel command sequence. |
| 10 | HC_INTERNAL_ERR_STAT | RW1C | 0x0 | Controller internal unrecoverable error. |
| 9:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.INTR_STATUS_ENABLE

*Table 2034. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_STATUS_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | Reserved | - | - | Reserved |
| 14 | SCHED_CMD_MISSED_TICK_STAT_EN | RW | 0x0 | Enable SCHED_CMD_MISSED_TICK_STAT monitoring. |
| 13 | HC_ERR_CMD_SEQ_TIMEOUT_STAT_EN | RW | 0x0 | Enable HC_ERR_CMD_SEQ_TIMEOUT_STAT monitoring. |
| 12 | HC_WARN_CMD_SEQ_STALL_STAT_EN | RW | 0x0 | Enable HC_WARN_CMD_SEQ_STALL_STAT monitoring. |
| 11 | HC_SEQ_CANCEL_STAT_EN | RW | 0x0 | Enable HC_SEQ_CANCEL_STAT monitoring. |
| 10 | HC_INTERNAL_ERR_STAT_EN | RW | 0x0 | Enable HC_INTERNAL_ERR_STAT monitoring. |
| 9:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE

*Table 2035. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_SIGNAL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | Reserved | - | - | Reserved |
| 14 | SCHED_CMD_MISSED_TICK_SIGNAL_EN | RW | 0x0 | Enable SCHED_CMD_MISSED_TICK_STAT interrupt. |
| 13 | HC_ERR_CMD_SEQ_TIMEOUT_SIGNAL_EN | RW | 0x0 | Enable HC_ERR_CMD_SEQ_TIMEOUT_STAT interrupt. |
| 12 | HC_WARN_CMD_SEQ_STALL_SIGNAL_EN | RW | 0x0 | Enable HC_WARN_CMD_SEQ_STALL_STAT interrupt. |
| 11 | HC_SEQ_CANCEL_SIGNAL_EN | RW | 0x0 | Enable HC_SEQ_CANCEL_STAT interrupt. |
| 10 | HC_INTERNAL_ERR_SIGNAL_EN | RW | 0x0 | Enable HC_INTERNAL_ERR_STAT interrupt. |
| 9:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.INTR_FORCE

*Table 2036. oca_i3c_wrap.i3c_csr[6].I3CBase.INTR_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | Reserved | - | - | Reserved |
| 14 | SCHED_CMD_MISSED_TICK_FORCE | W | 0x0 | Force SCHED_CMD_MISSED_TICK_STAT interrupt. |
| 13 | HC_ERR_CMD_SEQ_TIMEOUT_FORCE | W | 0x0 | Force HC_ERR_CMD_SEQ_TIMEOUT_STAT interrupt. |
| 12 | HC_WARN_CMD_SEQ_STALL_FORCE | W | 0x0 | Force HC_WARN_CMD_SEQ_STALL_STAT interrupt. |
| 11 | HC_SEQ_CANCEL_FORCE | W | 0x0 | Force HC_SEQ_CANCEL_STAT interrupt. |
| 10 | HC_INTERNAL_ERR_FORCE | W | 0x0 | Force HC_INTERNAL_ERR_STAT interrupt. |
| 9:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.DAT_SECTION_OFFSET

*Table 2037. oca_i3c_wrap.i3c_csr[6].I3CBase.DAT_SECTION_OFFSET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `ENTRY_SIZE` | R | 0x0 | Individual DAT entry size. 0 - 2 DWRODs, 1:15 - reserved. |
| 27:19 | `Reserved` | - | - | Reserved |
| 18:12 | `TABLE_SIZE` | R | 0x7F | Max number of DAT entries. |
| 11:0 | `TABLE_OFFSET` | R | 0x400 | DAT entry offset in respect to BASE address. |

### i3c_csr[6].I3CBase.DCT_SECTION_OFFSET

*Table 2038. oca_i3c_wrap.i3c_csr[6].I3CBase.DCT_SECTION_OFFSET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `ENTRY_SIZE` | R | 0x0 | Individual DCT entry size.<br><br>0 - 4 DWORDs,<br><br>1:15 - Reserved. |
| 27:24 | `Reserved` | - | - | Reserved |
| 23:19 | `TABLE_INDEX` | RW | 0x0 | Index to DCT used during ENTDAA. |
| 18:12 | `TABLE_SIZE` | R | 0x7F | Max number of DCT entries. |
| 11:0 | `TABLE_OFFSET` | R | 0x800 | DCT entry offset in respect to BASE address. |

### i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET

*Table 2039. oca_i3c_wrap.i3c_csr[6].I3CBase.RING_HEADERS_SECTION_OFFSET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `SECTION_OFFSET` | R | 0x0 | DMA ring headers section offset. Invalid if 0. |

### i3c_csr[6].I3CBase.PIO_SECTION_OFFSET

*Table 2040. oca_i3c_wrap.i3c_csr[6].I3CBase.PIO_SECTION_OFFSET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `SECTION_OFFSET` | R | 0x80 | PIO section offset. Invalid if 0. |

### i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET

*Table 2041. oca_i3c_wrap.i3c_csr[6].I3CBase.EXT_CAPS_SECTION_OFFSET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `SECTION_OFFSET` | R | 0x100 | Extended Capabilities section offset. Invalid if 0. |

### i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN

*Table 2042. oca_i3c_wrap.i3c_csr[6].I3CBase.INT_CTRL_CMDS_EN fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:1 | `MIPI_CMDS_SUPPORTED` | R | 0x35 | Bitmask of supported MIPI commands. |
| 0 | `ICC_SUPPORT` | R | 0x1 | Internal Control Commands:<br><br>1 - some or all internals commands sub-commands are supported,<br><br>0 - illegal. |

### i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL

*Table 2043. oca_i3c_wrap.i3c_csr[6].I3CBase.IBI_NOTIFY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `NOTIFY_IBI_REJECTED` | RW | 0x0 | Notify about rejected IBI:<br><br>0 - do not enqueue rejected IBI,<br><br>1 = enqueue rejected IBI on IBI queue/ring. |
| 2 | `Reserved` | - | - | Reserved |
| 1 | `NOTIFY_CRR_REJECTED` | RW | 0x0 | Notify about rejected controller role request:<br><br>0 - do not enqueue rejected CRR,<br><br>1 = enqueue rejected CRR on IBI queue/ring. |
| 0 | `NOTIFY_HJ_REJECTED` | RW | 0x0 | Notify about rejected hot-join:<br><br>0 - do not enqueue rejected HJ,<br><br>1 = enqueue rejected HJ on IBI queue/ring. |

### i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL

*Table 2044. oca_i3c_wrap.i3c_csr[6].I3CBase.IBI_DATA_ABORT_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `IBI_DATA_ABORT_MON` | RW | 0x0 | Enable/disable IBI monitoring logic. |
| 30:21 | `Reserved` | - | - | Reserved |
| 20:18 | `MATCH_STATUS_TYPE` | RW | 0x0 | Define which IBI should be aborted:<br><br>3’b000 - Regular IBI,<br><br>3’b100 - Autocmd IBI,<br><br>other values - not supported. |
| 17:16 | `AFTER_N_CHUNKS` | RW | 0x0 | Number of data chunks to be allowed before forced termination:<br><br>0 - immediate,<br><br>1:3 - delay by 1-3 data chunks. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:8 | MATCH_IBI_ID | RW | 0x0 | IBI target address:<br><br>[15:9] - device address,<br><br>[8] - must always be set to 1'b1 |
| 7:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3CBase.DEV_CTX_BASE_LO

*Table 2045. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_BASE_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | - | - | Reserved |
| 0 | BASE_LO | RW | 0x0 | - |

### i3c_csr[6].I3CBase.DEV_CTX_BASE_HI

*Table 2046. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_BASE_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | - | - | Reserved |
| 0 | BASE_HI | RW | 0x0 | - |

### i3c_csr[6].I3CBase.DEV_CTX_SG

*Table 2047. oca_i3c_wrap.i3c_csr[6].I3CBase.DEV_CTX_SG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | BLP | R | 0x0 | Buffer vs list pointer in device context:<br><br>0 - continuous physical memory region,<br><br>1 - pointer to SG descriptor list. |
| 30:16 | Reserved | - | - | Reserved |
| 15:0 | LIST_SIZE | R | 0x0 | Number of SG entries. |

### i3c_csr[6].PIOControl.COMMAND_PORT

*Table 2048. oca_i3c_wrap.i3c_csr[6].PIOControl.COMMAND_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | COMMAND_DATA | W | - | - |

### i3c_csr[6].PIOControl.RESPONSE_PORT

*Table 2049. oca_i3c_wrap.i3c_csr[6].PIOControl.RESPONSE_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RESPONSE_DATA | R | - | - |

### i3c_csr[6].PIOControl.TX_DATA_PORT

*Table 2050. oca_i3c_wrap.i3c_csr[6].PIOControl.TX_DATA_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | TX_DATA | W | - | - |

### i3c_csr[6].PIOControl.RX_DATA_PORT

*Table 2051. oca_i3c_wrap.i3c_csr[6].PIOControl.RX_DATA_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | RX_DATA | R | - | - |

### i3c_csr[6].PIOControl.IBI_PORT

*Table 2052. oca_i3c_wrap.i3c_csr[6].PIOControl.IBI_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | IBI_DATA | R | - | Data Read from IBI Data Buffer.<br><br>The IBI Port is mapped to the IBI Data Queue. IBI Data is always aligned to a 4-byte boundary and then put into the IBI Queue. If the incoming data is not aligned to a 4-byte boundary, then there will be extra (unused) bytes at the end of the transferred IBI data. This can be determined from the value of field DATA_LENGTH in the IBI Status Descriptor. |

### i3c_csr[6].PIOControl.QUEUE_THLD_CTRL

*Table 2053. oca_i3c_wrap.i3c_csr[6].PIOControl.QUEUE_THLD_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | IBI_STATUS_THLD | RW | 0x1 | Triggers IBI_STATUS_THLD_STAT interrupt when IBI queue has N or more entries. Accepted values are 1:255 |
| 23:16 | IBI_DATA_SEGMENT_SIZE | RW | 0x1 | IBI Queue data segment size. Valida values are 1:63 |
| 15:8 | RESP_BUF_THLD | RW | 0x1 | Triggers RESP_READY_STAT interrupt when RESP queue has N or more entries. Accepted values are 1:255 |
| 7:0 | CMD_EMPTY_BUF_THLD | RW | 0x1 | Triggers CMD_QUEUE_READY_STAT interrupt when CMD queue has N or more free entries. Accepted values are 1:255 |

### i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL

*Table 2054. oca_i3c_wrap.i3c_csr[6].PIOControl.DATA_BUFFER_THLD_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:27 | Reserved | - | - | Reserved |
| 26:24 | RX_START_THLD | RW | 0x1 | Postpone read command until RX queue has 2^(N+1) free entries |
| 23:19 | Reserved | - | - | Reserved |
| 18:16 | TX_START_THLD | RW | 0x1 | Postpone write command until TX queue has 2^(N+1) entries |
| 15:11 | Reserved | - | - | Reserved |
| 10:8 | RX_BUF_THLD | RW | 0x1 | Trigger RX_THLD_STAT interrupt when RX queue has 2^(N+1) or more entries |
| 7:3 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2:0 | TX_BUF_THLD | RW | 0x1 | Trigger TX_THLD_STAT interrupt when TX queue has 2^(N+1) or more free entries |

### i3c_csr[6].PIOControl.QUEUE_SIZE

*Table 2055. oca_i3c_wrap.i3c_csr[6].PIOControl.QUEUE_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | TX_DATA_BUFFER_SIZE | R | 0x5 | TX queue size is equal to 2^(N+1), where N is this field value |
| 23:16 | RX_DATA_BUFFER_SIZE | R | 0x5 | RX queue size is equal to 2^(N+1), where N is this field value |
| 15:8 | IBI_STATUS_SIZE | R | 0x20 | IBI Queue size is equal to N |
| 7:0 | CR_QUEUE_SIZE | R | 0x40 | Command/Response queue size is equal to N |

### i3c_csr[6].PIOControl.ALT_QUEUE_SIZE

*Table 2056. oca_i3c_wrap.i3c_csr[6].PIOControl.ALT_QUEUE_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | Reserved | - | - | Reserved |
| 28 | EXT_IBI_QUEUE_EN | R | 0x0 | 1 indicates that IBI queue size is equal to 8*IBI_STATUS_SIZE |
| 27:25 | Reserved | - | - | Reserved |
| 24 | ALT_RESP_QUEUE_EN | R | 0x1 | If set, response and command queues are not equal lengths, then ALT_RESP_QUEUE_SIZE contains response queue size |
| 23:8 | Reserved | - | - | Reserved |
| 7:0 | ALT_RESP_QUEUE_SIZE | R | 0x20 | Valid only if ALT_RESP_QUEUE_EN is set. Contains response queue size |

### i3c_csr[6].PIOControl.PIO_INTR_STATUS

*Table 2057. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9 | TRANSFER_ERR_STAT | RW1C | 0x0 | Transfer error |
| 8:6 | Reserved | - | - | Reserved |
| 5 | TRANSFER_ABORT_STAT | RW1C | 0x0 | Transfer aborted |
| 4 | RESP_READY_STAT | R | 0x0 | Response queue fulfils RESP_BUF_THLD |
| 3 | CMD_QUEUE_READY_STAT | R | 0x0 | Command queue fulfils CMD_EMPTY_BUF_THLD |
| 2 | IBI_STATUS_THLD_STAT | R | 0x0 | IBI queue fulfils IBI_STATUS_THLD |
| 1 | RX_THLD_STAT | R | 0x0 | RX queue fulfils RX_BUF_THLD |
| 0 | TX_THLD_STAT | R | 0x0 | TX queue fulfils TX_BUF_THLD |

### i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE

*Table 2058. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_STATUS_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | Reserved | - | - | Reserved |
| 9 | TRANSFER_ERR_STAT_EN | RW | 0x0 | Enable transfer error monitoring |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 8:6 | `Reserved` | - | - | Reserved |
| 5 | `TRANSFER_ABORT_STAT_EN` | RW | 0x0 | Enable transfer abort monitoring |
| 4 | `RESP_READY_STAT_EN` | RW | 0x0 | Enable response queue monitoring |
| 3 | `CMD_QUEUE_READY_STAT_EN` | RW | 0x0 | Enable command queue monitoring |
| 2 | `IBI_STATUS_THLD_STAT_EN` | RW | 0x0 | Enable IBI queue monitoring |
| 1 | `RX_THLD_STAT_EN` | RW | 0x0 | Enable RX queue monitoring |
| 0 | `TX_THLD_STAT_EN` | RW | 0x0 | Enable TX queue monitoring |

### i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE

*Table 2059. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_SIGNAL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9 | `TRANSFER_ERR_SIGNAL_EN` | RW | 0x0 | Enable transfer error interrupt |
| 8:6 | `Reserved` | - | - | Reserved |
| 5 | `TRANSFER_ABORT_SIGNAL_EN` | RW | 0x0 | Enable transfer abort interrupt |
| 4 | `RESP_READY_SIGNAL_EN` | RW | 0x0 | Enable response ready interrupt |
| 3 | `CMD_QUEUE_READY_SIGNAL_EN` | RW | 0x0 | Enable command queue interrupt |
| 2 | `IBI_STATUS_THLD_SIGNAL_EN` | RW | 0x0 | Enable IBI queue interrupt |
| 1 | `RX_THLD_SIGNAL_EN` | RW | 0x0 | Enable RX queue interrupt |
| 0 | `TX_THLD_SIGNAL_EN` | RW | 0x0 | Enable TX queue interrupt |

### i3c_csr[6].PIOControl.PIO_INTR_FORCE

*Table 2060. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_INTR_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9 | `TRANSFER_ERR_FORCE` | W | 0x0 | Force transfer error |
| 8:6 | `Reserved` | - | - | Reserved |
| 5 | `TRANSFER_ABORT_FORCE` | W | 0x0 | Force transfer aborted |
| 4 | `RESP_READY_FORCE` | W | 0x0 | Force response queue interrupt |
| 3 | `CMD_QUEUE_READY_FORCE` | W | 0x0 | Force command queue interrupt |
| 2 | `IBI_THLD_FORCE` | W | 0x0 | Force IBI queue interrupt |
| 1 | `RX_THLD_FORCE` | W | 0x0 | Force RX queue interrupt |
| 0 | `TX_THLD_FORCE` | W | 0x0 | Force TX queue interrupt |

### i3c_csr[6].PIOControl.PIO_CONTROL

*Table 2061. oca_i3c_wrap.i3c_csr[6].PIOControl.PIO_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `ABORT` | RW | 0x0 | Stop current command descriptor execution forcefully and hold remaining commands. 1 - Request PIO Abort, 0 - Resume PIO execution |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | RS | RW | 0x0 | Run/Stop execution of enqueued commands. When set to 0, it holds execution of enqueued commands and runs current command to completion. 1 - PIO Queue start request, 0 - PIO Queue stop request. |
| 0 | ENABLE | RW | 0x1 | Enables PIO queues. When disabled, SW may not read from/write to PIO queues. 1 - PIO queue enable request, 0 - PIO queue disable request |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER

*Table 2062. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |
| 23:8 | CAP_LENGTH | R | 0x20 | Capability Structure Length in DWORDs |
| 7:0 | CAP_ID | R | 0xC0 | Extended Capability ID |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0

*Table 2063. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | REC_MAGIC_STRING_0 | R | 0x2050434F | Magic string 'OCP ' (1st part of 'OCP RECV') in ASCII code - '0x4f 0x43 0x50 0x20' |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1

*Table 2064. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | REC_MAGIC_STRING_1 | R | 0x56434552 | Magic string 'RECV' (2nd part of 'OCP RECV') in ASCII code - '0x52 0x45 0x43 0x56' |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2

*Table 2065. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | AGENT_CAPS | RW | 0x0 | Agent capabilities:<br>• bit 0: Identification (DEVICE_ID structure)<br>• bit 1: Forced Recovery (From RESET)<br>• bit 2: Mgmt reset (From RESET)<br>• bit 3: Device Reset (From RESET)<br>• bit 4: Device status (DEVICE_STATUS)<br>• bit 5: Recovery memory access (INDIRECT_CTRL)<br>• bit 6: Local C-image support<br>• bit 7: Push C-image support<br>• bit 8: Interface isolation<br>• bit 9: Hardware status<br>• bit 10: Vendor command<br>• bit 11: Flashless boot (From RESET)<br>• bit 12: FIFO CMS support (INDIRECT_FIFO_CTRL)<br>• bits 13-15: Reserved |
| 15:0 | REC_PROT_VERSION | RW | 0x0 | • Byte 0: Major version number = 0x1<br>• Byte 1: Minor version number = 0x1 |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3

*Table 2066. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.PROT_CAP_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |
| 23:16 | HEARTBEAT_PERIOD | RW | 0x0 | 0-255: Heartbeat period, 2^x microseconds (us), 0 indicates not supported |
| 15:8 | MAX_RESP_TIME | RW | 0x0 | 0-255: Maximum response time in 2^x microseconds(us). |
| 7:0 | NUM_OF_CMS_REGIONS | RW | 0x0 | 0-255: The total number of component memory space (CMS) regions a device supports. This number includes any logging, code and vendor defined regions |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0

*Table 2067. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | DATA | RW | 0x0 | - |
| 15:8 | VENDOR_SPECIFIC_STR_LENGTH | RW | 0x0 | 0x0-0xFF: Total length of Vendor Specific String, 0 indicates not supported |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | DESC_TYPE | RW | 0x0 | Based on table 8 from [DMTF PLDM FM]:<br>• 0x00: PCI Vendor<br>• 0x1: IANA<br>• 0x2: UUID<br>• 0x3: PnP Vendor<br>• 0x4: ACPI Vendor<br>• 0x5: IANA Enterprise Type<br>• 0x6-0xFE: Reserved<br>• 0xFF: NVMe-MI |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1

*Table 2068. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | - |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2

*Table 2069. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | - |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3

*Table 2070. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | - |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4

*Table 2071. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | - |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5

*Table 2072. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | RW | 0x0 | - |

#### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED

*Table 2073. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_ID_RESERVED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | - |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0

*Table 2074. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | REC_REASON_CODE | RW | 0x0 | • 0x0: No Boot Failure detected (BFNF)<br>• 0x1: Generic hardware error (BFGHWE)<br>• 0x2: Generic hardware soft error (BFGSE) - soft error may be recoverable<br>• 0x3: Self-test failure (BFSTF) (e.g., RSA self test failure, FIPs self test failure,, etc.)<br>• 0x4: Corrupted/missing critical data (BFCD)<br>• 0x5: Missing/corrupt key manifest (BFKMMC)<br>• 0x6: Authentication Failure on key manifest (BFKMAF)<br>• 0x7: Anti-rollback failure on key manifest (BFKIAR)<br>• 0x8: Missing/corrupt boot loader (first mutable code) firmware image (BFFIMC)<br>• 0x9: Authentication failure on boot loader ( 1st mutable code) firmware image (BFFIAF)<br>• 0xA: Anti-rollback failure boot loader (1st mutable code) firmware image (BFFIAR)<br>• 0xB: Missing/corrupt main/management firmware image (BFMFMC)<br>• 0xC: Authentication Failure main/management firmware image (BFMFAF)<br>• 0xD: Anti-rollback Failure main/management firmware image (BFMFAR)<br>• 0xE: Missing/corrupt recovery firmware (BFRFMC)<br>• 0xF: Authentication Failure recovery firmware (BFRFAF)<br>• 0x10: Anti-rollback Failure on recovery firmware (BFRFAR)<br>• 0x11: Forced Recovery (FR)<br>• 0x12: Flashless/Streaming Boot (FSB)<br>• 0x13-0x7F: Reserved<br>• 0x80-0xFF: Vendor Unique Boot Failure Codes |
| 15:8 | PROT_ERROR | RW RC | 0x0 | • 0x0: No Protocol Error<br>• 0x1: Unsupported/Write Command - command is not support or a write to a RO command<br>• 0x2: Unsupported Parameter<br>• 0x3: Length write error (length of write command is incorrect)<br>• 0x4: CRC Error (if supported)<br>• 0x5-0xFE: Reserved<br>• 0xFF: General Protocol Error - catch all unclassified errors |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | DEV_STATUS | RW | 0x0 | • 0x0: Status Pending (Recover Reason Code not populated)<br>• 0x1: Device healthy (Recover Reason Code not populated)<br>• 0x2: Device Error (“soft” error or other error state) - (Recover Reason Code not populated)<br>• 0x3: Recovery mode - ready to accept recovery image - (Recover Reason Code populated)<br>• 0x4: Recovery Pending (waiting for activation) - (Recover Reason Code populated)<br>• 0x5: Running Recovery Image ( Recover Reason Code not populated)<br>• 0x6-0xD: Reserved<br>• 0xE: Boot Failure (Recover Reason Code populated)<br>• 0xF: Fatal Error (Recover Reason Code not populated)<br>• 0x10-FF:Reserved |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1

*Table 2075. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_STATUS_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | VENDOR_STATUS | RW | 0x0 | - |
| 24:16 | VENDOR_STATUS_LENGTH | RW | 0x0 | 0-248: Length in bytes of just VENDOR_STATUS. Zero indicates no vendor status and zero additional bytes. |
| 15:0 | HEARTBEAT | RW | 0x0 | 0-4095: Incrementing number (counter wraps) |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET

*Table 2076. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.DEVICE_RESET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |
| 23:16 | IF_CTRL | RW | 0x0 | • 0x0: Disable Interface mastering<br>• 0x1: Enable Interface mastering |
| 15:8 | FORCED_RECOVERY | RW | 0x0 | • 0x0: No forced recovery<br>• 0x01-0xD: Reserved<br>• 0xE: Enter flashless boot mode on next platform reset<br>• 0xF: Enter recovery mode on next platform reset<br>• 0x10-FF: Reserved |
| 7:0 | RESET_CTRL | RW1C | 0x0 | • 0x0: No reset<br>• 0x1: Reset Device (PCIe Fundamental Reset or equivalent. This is likely bus disruptive)<br>• 0x2: Reset Management. This reset will reset the management subsystem. If supported, this reset MUST not be bus disruptive (cause re-enumeration)<br>• 0x3-FF: Reserved |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL

*Table 2077. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:16 | `ACTIVATE_REC_IMG` | RW1C | 0x0 | • 0x0: do not activate recovery image - after activation device will report this code.<br>• 0xF: Activate recovery image<br>• 0x10-FF-reserved |
| 15:8 | `REC_IMG_SEL` | RW | 0x0 | • 0x0: No operation<br>• 0x1: Use Recovery Image from memory window (CMS)<br>• 0x2: Use Recovery Image stored on device (C-image)<br>• 0x3-FF: reserved |
| 7:0 | `CMS` | RW | 0x0 | • 0-255: Selects a component memory space where the recovery image is. 0 is the default |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS

*Table 2078. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.RECOVERY_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:8 | `VENDOR_SPECIFIC_STATUS` | RW | 0x0 | - |
| 7:4 | `REC_IMG_INDEX` | RW | 0x0 | - |
| 3:0 | `DEV_REC_STATUS` | RW | 0x0 | • 0x0: Not in recovery mode<br>• 0x1: Awaiting recovery image<br>• 0x2: Booting recovery image<br>• 0x3: Recovery successful<br>• 0xc: Recovery failed<br>• 0xd: Recovery image authentication error<br>• 0xe: Error entering Recovery mode (might be administratively disabled)<br>• 0xf: Invalid component address space |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS

*Table 2079. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.HW_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `VENDOR_HW_STATUS_LEN` | RW | 0x0 | 0-251: Length in bytes of Vendor Specific Hardware Status. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 23:16 | `CTEMP` | RW | 0x0 | Current temperatureof device in degrees Celsius: Compatible with NVMe-MI command code 0 offset 3.<br>• 0x00-0x7e: 0 to 126 C<br>• 0x7f: 127 C or higher<br>• 0x80: no temperature data, or data is older than 5 seconds<br>• 0x81: temperature sensor failure<br>• 0x82-0x83: reserved<br>• 0xc4: -60 C or lower<br>• 0xc5-0xff: -59 to -1 C (in two's complement) |
| 15:8 | `VENDOR_HW_STATUS` | RW | 0x0 | - |
| 7:3 | `RESERVED_7_3` | RW | 0x0 | - |
| 2 | `FATAL_ERR` | RW | 0x0 | - |
| 1 | `SOFT_ERR` | RW | 0x0 | Hardware Soft Error (may need reset to clear) |
| 0 | `TEMP_CRITICAL` | RW | 0x0 | Device temperature is critical (may need reset to clear) |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0

*Table 2080. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:8 | `RESET` | RW | 0x0 | Reset (Write 1 Clear):<br>• 0x0: idle<br>• 0x1: reset Write Index and Read Index to initial value.<br>• 0x2 to 0xFF: reserved |
| 7:0 | `CMS` | RW | 0x0 | This register selects a region within the device. Read/write access is through address spaces. Each space represents a FIFO. Component Memory Space (CMS):<br>• 0-255: Address region within a device. |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1

*Table 2081. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_CTRL_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `IMAGE_SIZE` | RW | 0x0 | Image Size: Size of the image to be loaded in 4B units |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0

*Table 2082. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:11 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 10:8 | `REGION_TYPE` | R | 0x0 | Memory Region Type:<br>• 0b000: Code space for recovery. (write only)<br>• 0b001: Log uses the defined debug format (read only)<br>• 0b100: Vendor Defined Region (write only)<br>• 0b101: Vendor Defined Region (read only)<br>• 0b111: Unsupported Region (address space out of range) |
| 7:2 | `Reserved` | - | - | Reserved |
| 1 | `FULL` | R | 0x0 | If set, FIFO is full |
| 0 | `EMPTY` | R | 0x1 | If set, FIFO is empty |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1

*Table 2083. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `WRITE_INDEX` | R | 0x0 | Offset incremented for each access by the Recovery Agent in 4B units |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2

*Table 2084. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `READ_INDEX` | R | 0x0 | Offset incremented for each access by the device in 4B units |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3

*Table 2085. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `FIFO_SIZE` | R | 0x40 | Size of memory window specified in 4B units. Current implementation supports only a constant size of 64. |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4

*Table 2086. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_STATUS_4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `MAX_TRANSFER_SIZE` | R | 0x40 | Max size of the data payload in each read/write to INDIRECT_FIFO_DATA in 4B units<br><br>Enforced to 256 bytes (64 DWORDs) by Caliptra Subsystem Recovery Sequence |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED

*Table 2087. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_RESERVED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `DATA` | R | 0x0 | - |

### i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA

*Table 2088. oca_i3c_wrap.i3c_csr[6].I3C_EC.SecFwRecoveryIf.INDIRECT_FIFO_DATA fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R | 0x0 | - |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER

*Table 2089. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |
| 23:8 | CAP_LENGTH | R | 0x20 | Capability Structure Length in DWORDs |
| 7:0 | CAP_ID | R | 0x12 | Extended Capability ID |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL

*Table 2090. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:30 | STBY_CR_ENABLE_INIT | RW | 0x0 | Enables or disables the Secondary Controller:<br><br>2'b00 - DISABLED: Secondary Controller is disabled.<br><br>2'b01 - ACM_INIT: Secondary Controller is enabled, but Host Controller initializes in Active Controller mode.<br><br>2'b10 - SCM_RUNNING: Secondary Controller operation is enabled, Host Controller initializes in Standby Controller mode.<br><br>2'b11 - SCM_HOT_JOIN: Secondary Controller operation is enabled, Host Controller conditionally becomes a Hot-Joining Device to receive its Dynamic Address before operating in Standby Controller mode. |
| 29:21 | Reserved | - | - | Reserved |
| 20 | RSTACT_DEFBYTE_02 | RW | 0x0 | Controls whether I3C Secondary Controller Logic supports RSTACT CCC with Defining Byte 0x02.<br><br>1'b0: NOT_SUPPORTED: Do not ACK Defining Byte 0x02<br><br>1'b1: HANDLE_INTR: Support Defining Byte 0x02 |
| 19:16 | Reserved | - | - | Reserved |
| 15 | DAA_ENTDAA_ENABLE | RW | 0x0 | Indicates ENTDAA method is enabled.<br><br>1'b0: DISABLED: will not respond<br><br>1'b1: ENABLED: will respond |
| 14 | DAA_SETDASA_ENABLE | RW | 0x0 | Indicates SETDASA method is enabled.<br><br>1'b0: DISABLED: will not respond<br><br>1'b1: ENABLED: will respond |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 13 | DAA_SETAASA_ENABLE | RW | 0x0 | Indicates SETAASA method is enabled.<br><br>1'b0: DISABLED: will not respond<br><br>1'b1: ENABLED: will respond |
| 12 | TARGET_XACT_ENABLE | RW | 0x1 | Indicates whether Read-Type/Write-Type transaction servicing is enabled, via an I3C Target Transaction Interface to software (Section 6.17.3).<br><br>1'b0: DISABLED: not available<br><br>1'b1: ENABLED: available for software |
| 11 | Reserved | - | - | Reserved |
| 10:8 | BAST_CCC_IBI_RING | RW | 0x0 | Indicates which Ring Bundle will be used to capture Broadcast CCC data sent by the Active Controller. The Ring Bundle must be configured and enabled, and its IBI Ring Pair must also be initialized and ready to receive data. |
| 7:6 | Reserved | - | - | Reserved |
| 5 | CR_REQUEST_SEND | W | 0x0 | Write of 1'b1 to this field shall instruct the Secondary Controller Logic to attempt to send a Controller Role Request to the I3C Bus. |
| 4 | HANDOFF_DEEP_SLEEP | RW WS | 0x0 | If this field has a value of 1'b1, then the Secondary Controller Logic shall report a return from Deep Sleep state to the Active Controller. Writing 1'b1 to this bit is sticky. This field shall automatically clear to 1'b0 after accepting the Controller Role and transitioning to Active Controller mode. |
| 3 | PRIME_ACCEPT_GETACCCR | RW | 0x0 | - |
| 2 | ACR_FSM_OP_SELECT | RW | 0x0 | - |
| 1 | HANDOFF_DELAY_NACK | RW | 0x0 | - |
| 0 | PENDING_RX_NACK | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR

*Table 2091. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | DYNAMIC_ADDR_VALID | RW | 0x0 | Indicates whether or not the value in the DYNAMIC_ADDR field is valid. 1'b0: DYNAMIC_ADDR field is not valid 1'b1: DYNAMIC_ADDR field is valid |
| 30:23 | Reserved | - | - | Reserved |
| 22:16 | DYNAMIC_ADDR | RW | 0x0 | Contains the Host Controller Device's Dynamic Address. |
| 15 | STATIC_ADDR_VALID | RW | 0x0 | Indicates whether or not the value in the STATIC_ADDR field is valid.<br><br>1'b0: The Static Address field is not valid<br><br>1'b1: The Static Address field is valid |
| 14:7 | Reserved | - | - | Reserved |
| 6:0 | STATIC_ADDR | RW | 0x0 | This field contains the Host Controller Device's Static Address. |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES

*Table 2092. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CAPABILITIES fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15 | DAA_ENTDAA_SUPPORT | R | 0x0 | Defines whether Dynamic Address Assignment with ENTDAA CCC is supported.<br><br>1'b0: DISABLED: Not supported<br><br>1'b1: ENABLED: Supported |
| 14 | DAA_SETDASA_SUPPORT | R | 0x1 | Defines whether Dynamic Address Assignment with SETDASA CCC (using Static Address) is supported.<br><br>1'b0: DISABLED: Not supported<br><br>1'b1: ENABLED: Supported |
| 13 | DAA_SETAASA_SUPPORT | R | 0x1 | Defines whether Dynamic Address Assignment with SETAASA CCC (using Static Address) is supported.<br><br>1'b0: DISABLED: Not supported<br><br>1'b1: ENABLED: Supported |
| 12 | TARGET_XACT_SUPPORT | R | 0x1 | Defines whether an I3C Target Transaction Interface is supported.<br><br>1'b0: DISABLED: Not supported<br><br>1'b1: ENABLED: Supported via vendor-defined Extended Capability structure |
| 11:6 | Reserved | - | - | Reserved |
| 5 | SIMPLE_CRR_SUPPORT | R | 0x0 | - |
| 4:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR

*Table 2093. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_CHAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | BCR_FIXED | RW | 0x1 | Bus Characteristics, Fixed Part.<br><br>Reset value is set to 3'b001, because this device is an I3C Target, which supports extended capabilities.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 28:24 | BCR_VAR | RW | 0x10 | Bus Characteristics, Variable Part.<br><br>Reset value is set to 5'b10000, because this device:<br><br>• [bit4] is a Virtual Target<br>• [bit3] is not Offline Capable<br>• [bit2] does not apply due to bit1<br>• [bit1] is not capable of IBI requests<br>• [bit0] has no speed limitation<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 23:16 | DCR | RW | 0xBD | Device Characteristics Register. Value represents an OCP Recovery Device.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 15:1 | PID_HI | RW | 0x7FFF | High part of the 48-bit Target Device Provisioned ID.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS

*Table 2094. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | Reserved | - | - | Reserved |
| 8 | HJ_REQ_STATUS | RW | 0x0 | - |
| 7:5 | SIMPLE_CRR_STATUS | RW | 0x0 | - |
| 4:3 | Reserved | - | - | Reserved |
| 2 | AC_CURRENT_OWN | RW | 0x0 | - |
| 1:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR

*Table 2095. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_CHAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | BCR_FIXED | RW | 0x1 | Bus Characteristics, Fixed Part.<br><br>Reset value is set to 3’b001, because this device is an I3C Target, which supports extended capabilities<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 28:24 | BCR_VAR | RW | 0x16 | Bus Characteristics, Variable Part.<br><br>Reset value is set to 5’b10110, because this device:<br><br>• [bit4] exposes a Virtual Target<br>• [bit3] is not Offline Capable<br>• [bit2] (always) uses the MDB in the IBI Payload<br>• [bit1] is capable of IBI requests<br>• [bit0] has no speed limitation<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 23:16 | DCR | RW | 0xBD | Device Characteristics Register. Value represents an OCP Recovery Device.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:1 | PID_HI | RW | 0x7FFF | High part of the 48-bit Target Device Provisioned ID.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |
| 0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO

*Table 2096. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_DEVICE_PID_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PID_LO | RW | 0x5A00A5 | Low part of the 48-bit Target Device Provisioned ID.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS

*Table 2097. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | CCC_FATAL_RSTDAA_ERR_STAT | RW | 0x0 | - |
| 18 | CCC_UNHANDLED_NACK_STAT | RW | 0x0 | - |
| 17 | CCC_PARAM_MODIFIED_STAT | RW | 0x0 | - |
| 16 | STBY_CR_OP_RSTACT_STAT | RW | 0x0 | The Host Controller shall write 1’b1 to this field to indicate that the Secondary Controller received a RSTACT CCC from the Active Controller, followed by the Target Reset Pattern. |
| 15 | Reserved | - | - | Reserved |
| 14 | STBY_CR_ACCEPT_ERR_STAT | RW | 0x0 | - |
| 13 | STBY_CR_ACCEPT_OK_STAT | RW | 0x0 | - |
| 12 | STBY_CR_ACCEPT_NACKED_STAT | RW | 0x0 | - |
| 11 | STBY_CR_DYN_ADDR_STAT | RW | 0x0 | - |
| 10 | CRR_RESPONSE_STAT | RW | 0x0 | - |
| 9:4 | Reserved | - | - | Reserved |
| 3 | ACR_HANDOFF_ERR_M3_STAT | RW | 0x0 | - |
| 2 | ACR_HANDOFF_ERR_FAIL_STAT | RW | 0x0 | - |
| 1 | ACR_HANDOFF_OK_PRIMED_STAT | RW | 0x0 | - |
| 0 | ACR_HANDOFF_OK_REMAIN_STAT | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO

*Table 2098. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRTUAL_DEVICE_PID_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PID_LO | RW | 0x5A10A5 | Low part of the 48-bit Target Virtual Device Provisioned ID.<br><br>This field is Read-Only while the STBY_CR_ENABLE_INIT field in register STBY_CR_CONTROL is not zero. |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE

*Table 2099. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_SIGNAL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | CCC_FATAL_RSTDAA_ERR_SIGNAL_EN | RW | 0x0 | - |
| 18 | CCC_UNHANDLED_NACK_SIGNAL_EN | RW | 0x0 | - |
| 17 | CCC_PARAM_MODIFIED_SIGNAL_EN | RW | 0x0 | - |
| 16 | STBY_CR_OP_RSTACT_SIGNAL_EN | RW | 0x0 | - |
| 15 | Reserved | - | - | Reserved |
| 14 | STBY_CR_ACCEPT_ERR_SIGNAL_EN | RW | 0x0 | - |
| 13 | STBY_CR_ACCEPT_OK_SIGNAL_EN | RW | 0x0 | - |
| 12 | STBY_CR_ACCEPT_NACKED_SIGNAL_EN | RW | 0x0 | - |
| 11 | STBY_CR_DYN_ADDR_SIGNAL_EN | RW | 0x0 | - |
| 10 | CRR_RESPONSE_SIGNAL_EN | RW | 0x0 | - |
| 9:4 | Reserved | - | - | Reserved |
| 3 | ACR_HANDOFF_ERR_M3_SIGNAL_EN | RW | 0x0 | - |
| 2 | ACR_HANDOFF_ERR_FAIL_SIGNAL_EN | RW | 0x0 | - |
| 1 | ACR_HANDOFF_OK_PRIMED_SIGNAL_EN | RW | 0x0 | - |
| 0 | ACR_HANDOFF_OK_REMAIN_SIGNAL_EN | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE

*Table 2100. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_INTR_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19 | CCC_FATAL_RSTDAA_ERR_FORCE | RW | 0x0 | - |
| 18 | CCC_UNHANDLED_NACK_FORCE | RW | 0x0 | - |
| 17 | CCC_PARAM_MODIFIED_FORCE | RW | 0x0 | - |
| 16 | STBY_CR_OP_RSTACT_FORCE | W | 0x0 | - |
| 15 | Reserved | - | - | Reserved |
| 14 | STBY_CR_ACCEPT_ERR_FORCE | RW | 0x0 | - |
| 13 | STBY_CR_ACCEPT_OK_FORCE | RW | 0x0 | - |
| 12 | STBY_CR_ACCEPT_NACKED_FORCE | RW | 0x0 | - |
| 11 | STBY_CR_DYN_ADDR_FORCE | RW | 0x0 | - |
| 10 | CRR_RESPONSE_FORCE | RW | 0x0 | - |
| 9:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS

*Table 2101. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_GETCAPS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | Reserved | - | - | Reserved |
| 11:8 | F2_CRCAP2_DEV_INTERACT | RW | 0x0 | - |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:3 | Reserved | - | - | Reserved |
| 2:0 | F2_CRCAP1_BUS_CONFIG | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS

*Table 2102. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_CCC_CONFIG_RSTACT_PARAMS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | RESET_DYNAMIC_ADDR | RW | 0x1 | If set to 1'b1, then the Secondary Controller Logic must clear its Dynamic Address in register STBY_CR_DEVICE_ADDR after receiving a Target Reset Pattern that followed a Broadcast or Direct SET RSTACT CCC sent to the Dynamic Address, with Defining Byte 0x01 or 0x02. Requires support for Dynamic Address Assignment with at least one supported method, such as the ENTDAA CCC, with field DAA_ENTDAA_ENABLE set to 1'b1 in register STBY_CR_CONTROL. If field ACR_FSM_OP_SELECT in register STBY_CR_CONTROL is set to 1'b1, then this field shall be cleared (i.e., readiness to accept the Controller Role shall be revoked) with this Target Reset Pattern. |
| 30:24 | Reserved | - | - | Reserved |
| 23:16 | RESET_TIME_TARGET | RW | 0x0 | For Direct GET CCC, this field is returned for Defining Byte 0x82. |
| 15:8 | RESET_TIME_PERIPHERAL | RW | 0x0 | For Direct GET CCC, this field is returned for Defining Byte 0x81. |
| 7:0 | RST_ACTION | R | 0x1 | Contains the Defining Byte received with the last Direct SET CCC sent by the Active Controller. If not armed (START or RSTACT with unsupported defining byte) set to spec default 0x1. |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR

*Table 2103. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_VIRT_DEVICE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | VIRT_DYNAMIC_ADDR_VALID | RW | 0x0 | Indicates whether or not the value in the VIRT_DYNAMIC_ADDR field is valid. 1'b0: VIRT_DYNAMIC_ADDR field is not valid 1'b1: VIRT_DYNAMIC_ADDR field is valid |
| 30:23 | Reserved | - | - | Reserved |
| 22:16 | VIRT_DYNAMIC_ADDR | RW | 0x0 | Contains the Controller Virtual Device's Dynamic Address. |
| 15 | VIRT_STATIC_ADDR_VALID | RW | 0x0 | Indicates whether or not the value in the VIRT_STATIC_ADDR field is valid.<br><br>1'b0: The Virtual Device Static Address field is not valid<br><br>1'b1: The Virtual Device Static Address field is valid |
| 14:7 | Reserved | - | - | Reserved |
| 6:0 | VIRT_STATIC_ADDR | RW | 0x0 | This field contains the Host Controller Device's Static Address. |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL

*Table 2104. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MWL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | Reserved | - | - | Reserved |
| 15:0 | MWL | R | 0x100 | This field contains the maximum write length (MWL) (read-only) |

### i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL

*Table 2105. oca_i3c_wrap.i3c_csr[6].I3C_EC.StdbyCtrlMode.STBY_CR_MRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:16 | `IBIL` | R | 0x10 | This field contains the IBI payload size (IBIL) (read-only) |
| 15:0 | `MRL` | R | 0x100 | This field contains the maximum read length (MRL) (read-only) |

### i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER

*Table 2106. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:8 | `CAP_LENGTH` | R | 0x40 | Capability Structure Length in DWORDs |
| 7:0 | `CAP_ID` | R | 0xC4 | Extended Capability ID |

### i3c_csr[6].I3C_EC.TTI.CONTROL

*Table 2107. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:13 | `IBI_RETRY_NUM` | RW | 0x0 | Number of times the Target Device will try to request an IBI before giving up.<br><br>Values:<br><br>0x0 - Device will never retry.<br><br>0x1-0x6 - Device will retry this many times.<br><br>0x7 - Device will retry indefinitely until the Active Controller sets DISINT bit in the DISEC command. |
| 12 | `IBI_EN` | RW | 0x1 | Enable the IBI queue servicing.<br><br>Values:<br><br>0x0 - Device will not service the IBI queue.<br><br>0x1 - Device will send IBI requests onto the bus, if possible. |
| 11 | `CRR_EN` | RW | 0x0 | Enable Controller Role Request.<br><br>Values:<br><br>0x0 - Device is allowed to perform Controller Role Request.<br><br>0x1 - Device is not allowed to perform Controller Role Request. |
| 10 | `HJ_EN` | RW | 0x1 | Enable Hot-Join capability.<br><br>Values:<br><br>0x0 - Device is allowed to attempt Hot-Join.<br><br>0x1 - Device is not allowed to attempt Hot-Join. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 9:0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TTI.STATUS

*Table 2108. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:15 | `Reserved` | - | - | Reserved |
| 14:12 | `LAST_IBI_STATUS` | R | 0x0 | Status of last IBI. Should be read after IBI_DONE interrupt.<br><br>Values:<br><br>000 - Success: IBI was transmitted and ACK'd by the Active Controller. 001 - FailureNack: Active Controller NACK'd the IBI before any data was sent. The device will retry sending the IBI as long as permitted by the retry counter. 010 - FailurePartialData: Active Controller NACK'd the IBI after data was partially sent. The remaining part of data in the IBI queue is considered corrupted and will be discarded. 011 - FailureRetry: IBI could not be serviced due to reached retry count. 100 - FailureArbitration:: IBI could not be serviced as arbitration was lost during the address header. The device will retry sending the IBI as long as permitted by the retry counter. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `PROTOCOL_ERROR` | R | 0x0 | Protocol error occurred in the past. This field can only be reset by the Controller, if it issues the GETSTATUS CCC.<br><br>Values:<br><br>0 - no error occurred<br><br>1 - generic protocol error occurred in the past. It will be set until reception of the next GETSTATUS command. |
| 7:0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TTI.RESET_CONTROL

*Table 2109. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RESET_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:7 | `Reserved` | - | - | Reserved |
| 6 | `IBI_RETRY_CTR_RST` | W 1P | 0x0 | TTI IBI Retry Counter Software Reset |
| 5 | `IBI_QUEUE_RST` | RW | 0x0 | TTI IBI Queue Buffer Software Reset |
| 4 | `RX_DATA_RST` | RW | 0x0 | TTI RX Data Queue Buffer Software Reset |
| 3 | `TX_DATA_RST` | RW | 0x0 | TTI TX Data Queue Buffer Software Reset. Also resets the tti_conv_Nto8 since first dword is immediately loaded into the converter and can't be cleared otherwise. |
| 2 | `RX_DESC_RST` | RW | 0x0 | TTI RX Descriptor Queue Buffer Software Reset |
| 1 | `TX_DESC_RST` | RW | 0x0 | TTI TX Descriptor Queue Buffer Software Reset |
| 0 | `SOFT_RST` | RW | 0x0 | Target Core Software Reset |

### i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS

*Table 2110. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:10 | `Reserved` | - | - | Reserved |
| 9 | `IBI_QUEUE_EMPTY` | R | 0x1 | IBI Queue is empty |
| 8 | `IBI_QUEUE_FULL` | R | 0x0 | IBI Queue is full |
| 7 | `TX_DATA_QUEUE_EMPTY` | R | 0x1 | TX Data Queue is empty |
| 6 | `TX_DATA_QUEUE_FULL` | R | 0x0 | TX Data Queue is full |
| 5 | `RX_DATA_QUEUE_EMPTY` | R | 0x1 | RX Data Queue is empty |
| 4 | `RX_DATA_QUEUE_FULL` | R | 0x0 | RX Data Queue is full |
| 3 | `TX_DESC_QUEUE_EMPTY` | R | 0x1 | TX Descriptor Queue is empty |
| 2 | `TX_DESC_QUEUE_FULL` | R | 0x0 | TX Descriptor Queue is full |
| 1 | `RX_DESC_QUEUE_EMPTY` | R | 0x1 | RX Descriptor Queue is empty |
| 0 | `RX_DESC_QUEUE_FULL` | R | 0x0 | RX Descriptor Queue is full |

### i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH

*Table 2111. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DESC_QUEUE_DEPTH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:8 | `TX_DESC_QUEUE_DEPTH` | R | 0x0 | Number of entries in TX Descriptor Queue |
| 7:0 | `RX_DESC_QUEUE_DEPTH` | R | 0x0 | Number of entries in RX Descriptor Queue |

### i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH

*Table 2112. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DATA_QUEUE_DEPTH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:8 | `TX_DATA_QUEUE_DEPTH` | R | 0x0 | Number of DWORD entries in TX Data Queue |
| 7:0 | `RX_DATA_QUEUE_DEPTH` | R | 0x0 | Number of DWORD entries in RX Data Queue |

### i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH

*Table 2113. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_DEPTH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `IBI_QUEUE_DEPTH` | R | 0x0 | Number of entries in IBI Queue |

### i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS

*Table 2114. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `TRANSFER_ERR_STAT` | RW1C | 0x0 | NOT IMPLEMENTED. Bus error occurred. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 30:27 | `Reserved` | - | - | Reserved |
| 26 | `TX_DESC_COMPLETE` | RW1C | 0x0 | Read Transaction on the I3C Bus completed. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 25 | TRANSFER_ABORT_STAT | RW1C | 0x0 | NOT IMPLEMENTED. Bus aborted transaction. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 24:21 | Reserved | - | - | Reserved |
| 20 | PENDING_IBI | R | 0x0 | At least one IBI is pending. Read-only, hardware-driven level status: it tracks the live IBI-pending condition and cannot be cleared by software (clears automatically when no IBI is pending). |
| 19:16 | PENDING_INTERRUPT | RW | 0x0 | Contains the interrupt number of any pending interrupt, or 0 if no interrupts are pending. This encoding allows for up to 15 numbered interrupts. If more than one interrupt is set, then the highest priority interrupt shall be returned. Read-only, hardware-driven level status: it cannot be cleared by software and updates as pending conditions change. |
| 15:14 | Reserved | - | - | Reserved |
| 13 | IBI_DONE | RW1C | 0x0 | IBI is done, check LAST_IBI_STATUS for result. Clear: auto-cleared by hardware when software reads STATUS.LAST_IBI_STATUS; writing 1 to this bit in INTERRUPT_STATUS also clears it. |
| 12 | IBI_THLD_STAT | RW1C | 0x0 | TTI IBI Buffer Threshold Status, the Target Controller shall set this bit to 1 when the number of available entries in the TTI IBI Queue is >= the value defined in TTI_IBI_THLD. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 11 | RX_DESC_THLD_STAT | RW1C | 0x0 | TTI RX Descriptor Buffer Threshold Status, the Target Controller shall set this bit to 1 when the number of available entries in the TTI RX Descriptor Queue is >= the value defined in TTI_RX_DESC_THLD. Clear: auto-cleared by hardware when software reads the RX Descriptor Queue; writing 1 to this bit in INTERRUPT_STATUS also clears it. |
| 10 | TX_DESC_THLD_STAT | RW1C | 0x0 | TTI TX Descriptor Buffer Threshold Status, the Target Controller shall set this bit to 1 when the number of available entries in the TTI TX Descriptor Queue is >= the value defined in TTI_TX_DESC_THLD. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 9 | RX_DATA_THLD_STAT | RW1C | 0x0 | TTI RX Data Buffer Threshold Status, the Target Controller shall set this bit to 1 when the number of entries in the TTI RX Data Queue is >= the value defined in TTI_RX_DATA_THLD. Clear: auto-cleared by hardware when software reads the RX Data Queue; writing 1 to this bit in INTERRUPT_STATUS also clears it. |
| 8 | TX_DATA_THLD_STAT | RW1C | 0x0 | TTI TX Data Buffer Threshold Status, the Target Controller shall set this bit to 1 when the number of available entries in the TTI TX Data Queue is >= the value defined in TTI_TX_DATA_THLD. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 7:4 | Reserved | - | - | Reserved |
| 3 | TX_DESC_TIMEOUT | RW1C | 0x0 | NOT IMPLEMENTED. Pending Read was NACK'ed, because the TX_DESC_STAT event was not handled in time. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 2 | RX_DESC_TIMEOUT | RW1C | 0x0 | NOT IMPLEMENTED. Pending Write was NACK'ed, because the RX_DESC_STAT event was not handled in time. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear). |
| 1 | TX_DESC_STAT | RW1C | 0x0 | There is a pending Read Transaction on the I3C Bus. Software should write data to the TX Descriptor Queue and the TX Data Queue. Clear: write 1 to this bit in INTERRUPT_STATUS (no hardware auto-clear — writing to the TX queues does not clear it). |
| 0 | RX_DESC_STAT | RW1C | 0x0 | There is a pending Write Transaction. Software should read data from the RX Descriptor Queue and the RX Data Queue. Clear: auto-cleared by hardware when software reads the RX Descriptor Queue; writing 1 to this bit in INTERRUPT_STATUS also clears it. |

### i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE

*Table 2115. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | TRANSFER_ERR_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TRANSFER_ERR_STAT |
| 30:27 | Reserved | - | - | Reserved |
| 26 | TX_DESC_COMPLETE_EN | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_COMPLETE_EN |
| 25 | TRANSFER_ABORT_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TRANSFER_ABORT_STAT |
| 24:14 | Reserved | - | - | Reserved |
| 13 | IBI_DONE_EN | RW | 0x0 | Enables the corresponding interrupt bit IBI_DONE |
| 12 | IBI_THLD_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TTI_IBI_THLD_STAT |
| 11 | RX_DESC_THLD_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TTI_RX_DESC_THLD_STAT |
| 10 | TX_DESC_THLD_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TTI_TX_DESC_THLD_STAT |
| 9 | RX_DATA_THLD_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TTI_RX_DATA_THLD_STAT |
| 8 | TX_DATA_THLD_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TTI_TX_DATA_THLD_STAT |
| 7:4 | Reserved | - | - | Reserved |
| 3 | TX_DESC_TIMEOUT_EN | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_TIMEOUT_EN |
| 2 | RX_DESC_TIMEOUT_EN | RW | 0x0 | Enables the corresponding interrupt bit RX_DESC_TIMEOUT_EN |
| 1 | TX_DESC_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_STAT_EN |
| 0 | RX_DESC_STAT_EN | RW | 0x0 | Enables the corresponding interrupt bit RX_DESC_STAT_EN |

### i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE

*Table 2116. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.INTERRUPT_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | TRANSFER_ERR_STAT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit TRANSFER_ERR_STAT_FORCE |
| 30:27 | Reserved | - | - | Reserved |
| 26 | TX_DESC_COMPLETE_FORCE | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_COMPLETE_FORCE |
| 25 | TRANSFER_ABORT_STAT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit TRANSFER_ABORT_STAT_FORCE |
| 24:14 | Reserved | - | - | Reserved |
| 13 | IBI_DONE_FORCE | RW | 0x0 | Enables the corresponding interrupt bit IBI_DONE_FORCE |
| 12 | IBI_THLD_FORCE | RW | 0x0 | Forces the corresponding interrupt bit TTI_IBI_THLD_STAT to be set to 1 |
| 11 | RX_DESC_THLD_FORCE | RW | 0x0 | Forces the corresponding interrupt bit TTI_RX_DESC_THLD_STAT to be set to 1 |
| 10 | TX_DESC_THLD_FORCE | RW | 0x0 | Forces the corresponding interrupt bit TTI_TX_DESC_THLD_STAT to be set to 1 |
| 9 | RX_DATA_THLD_FORCE | RW | 0x0 | Forces the corresponding interrupt bit TTI_RX_DATA_THLD_STAT to be set to 1 |
| 8 | TX_DATA_THLD_FORCE | RW | 0x0 | Forces the corresponding interrupt bit TTI_TX_DATA_THLD_STAT to be set to 1 |
| 7:4 | Reserved | - | - | Reserved |
| 3 | TX_DESC_TIMEOUT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_TIMEOUT_FORCE |
| 2 | RX_DESC_TIMEOUT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit RX_DESC_TIMEOUT_FORCE |
| 1 | TX_DESC_STAT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit TX_DESC_STAT_FORCE |
| 0 | RX_DESC_STAT_FORCE | RW | 0x0 | Enables the corresponding interrupt bit RX_DESC_STAT_FORCE |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL

*Table 2117. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12 | `RI_INDIRECT_FIFO_OVERFLOW_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface INDIRECT FIFO overflow error detection and transition to Error state. Set to 0 to disable (overflow is still recorded in interrupt status). |
| 11 | `RI_RX_FIFO_OVERFLOW_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface RX FIFO overflow error detection and transition to Error state. Set to 0 to disable (overflow is still recorded in interrupt status). |
| 10 | `RI_UNSUPPORTED_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface unsupported command error detection. Set to 0 to disable. |
| 9 | `RI_READONLY_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface write-to-read-only error detection. Set to 0 to disable. |
| 8 | `RI_LENGTH_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface length error detection. Set to 0 to disable. |
| 7 | `RI_PEC_ERR_DET_EN` | RW | 0x1 | Enable Recovery Interface PEC/CRC error detection. Set to 0 to disable. |
| 6 | `FRAMING_ERR_DET_EN` | RW | 0x1 | Enable framing error detection. Set to 0 to disable. |
| 5 | `TE5_ERR_DET_EN` | RW | 0x1 | Enable TE5 error detection. Set to 0 to disable. |
| 4 | `TE4_ERR_DET_EN` | RW | 0x1 | Enable TE4 error detection. Set to 0 to disable. |
| 3 | `TE3_ERR_DET_EN` | RW | 0x1 | Enable TE3 error detection. Set to 0 to disable. |
| 2 | `TE2_ERR_DET_EN` | RW | 0x1 | Enable TE2 error detection. Set to 0 to disable. |
| 1 | `TE1_ERR_DET_EN` | RW | 0x1 | Enable TE1 error detection. Set to 0 to disable. |
| 0 | `TE0_ERR_DET_EN` | RW | 0x1 | Enable TE0 error detection. Set to 0 to disable. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS

*Table 2118. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:14 | `Reserved` | - | - | Reserved |
| 13 | `RI_INDIRECT_FIFO_OVERFLOW_ERR_STAT` | RW1C | 0x0 | Recovery Interface INDIRECT FIFO overflow error detected |
| 12 | `RI_RX_FIFO_OVERFLOW_ERR_STAT` | RW1C | 0x0 | Recovery Interface RX FIFO overflow error detected |
| 11 | `RI_UNSUPPORTED_ERR_STAT` | RW1C | 0x0 | Recovery Interface unsupported command error detected |
| 10 | `RI_READONLY_ERR_STAT` | RW1C | 0x0 | Recovery Interface write-to-read-only error detected |
| 9 | `RI_LENGTH_ERR_STAT` | RW1C | 0x0 | Recovery Interface length mismatch error detected |
| 8 | `RI_PEC_ERR_STAT` | RW1C | 0x0 | Recovery Interface PEC/CRC error detected |
| 7 | `FRAMING_ERR_STAT` | RW1C | 0x0 | DA padding error (Bit[0] != 0 in SETDASA/SETNEWDA) |
| 6 | `TE5_ERR_STAT` | RW1C | 0x0 | TE5: Broadcast/Direct CCC wrong R/W direction |
| 5 | `TE4_ERR_STAT` | RW1C | 0x0 | TE4: ENTDAA BCR/DCR mismatch |
| 4 | `TE3_ERR_STAT` | RW1C | 0x0 | TE3: ENTDAA PID mismatch |
| 3 | `TE2_ERR_STAT` | RW1C | 0x0 | TE2: CCC or Private Write data parity error |
| 2 | `TE1_ERR_STAT` | RW1C | 0x0 | TE1: CCC command parity error |
| 1 | `TE0_ERR_STAT` | RW1C | 0x0 | TE0: Invalid reserved address + RnW combination |
| 0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE

*Table 2119. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:14 | `Reserved` | - | - | Reserved |
| 13 | `RI_INDIRECT_FIFO_OVERFLOW_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_INDIRECT_FIFO_OVERFLOW_ERR_STAT` |
| 12 | `RI_RX_FIFO_OVERFLOW_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_RX_FIFO_OVERFLOW_ERR_STAT` |
| 11 | `RI_UNSUPPORTED_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_UNSUPPORTED_ERR_STAT` |
| 10 | `RI_READONLY_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_READONLY_ERR_STAT` |
| 9 | `RI_LENGTH_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_LENGTH_ERR_STAT` |
| 8 | `RI_PEC_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `RI_PEC_ERR_STAT` |
| 7 | `FRAMING_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `FRAMING_ERR_STAT` |
| 6 | `TE5_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE5_ERR_STAT` |
| 5 | `TE4_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE4_ERR_STAT` |
| 4 | `TE3_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE3_ERR_STAT` |
| 3 | `TE2_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE2_ERR_STAT` |
| 2 | `TE1_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE1_ERR_STAT` |
| 1 | `TE0_ERR_EN` | RW | 0x0 | Enables the corresponding interrupt bit `TE0_ERR_STAT` |
| 0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE

*Table 2120. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_INTR_FORCE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:14 | `Reserved` | - | - | Reserved |
| 13 | `RI_INDIRECT_FIFO_OVERFLOW_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_INDIRECT_FIFO_OVERFLOW_ERR_STAT` to be set to 1 |
| 12 | `RI_RX_FIFO_OVERFLOW_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_RX_FIFO_OVERFLOW_ERR_STAT` to be set to 1 |
| 11 | `RI_UNSUPPORTED_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_UNSUPPORTED_ERR_STAT` to be set to 1 |
| 10 | `RI_READONLY_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_READONLY_ERR_STAT` to be set to 1 |
| 9 | `RI_LENGTH_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_LENGTH_ERR_STAT` to be set to 1 |
| 8 | `RI_PEC_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `RI_PEC_ERR_STAT` to be set to 1 |
| 7 | `FRAMING_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `FRAMING_ERR_STAT` to be set to 1 |
| 6 | `TE5_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE5_ERR_STAT` to be set to 1 |
| 5 | `TE4_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE4_ERR_STAT` to be set to 1 |
| 4 | `TE3_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE3_ERR_STAT` to be set to 1 |
| 3 | `TE2_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE2_ERR_STAT` to be set to 1 |
| 2 | `TE1_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE1_ERR_STAT` to be set to 1 |
| 1 | `TE0_ERR_FORCE` | RW | 0x0 | Forces the corresponding interrupt bit `TE0_ERR_STAT` to be set to 1 |
| 0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0

*Table 2121. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1

*Table 2122. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2

*Table 2123. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3

*Table 2124. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4

*Table 2125. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE4 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5

*Table 2126. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_TE5 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING

*Table 2127. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_FRAMING fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC

*Table 2128. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_PEC fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH

*Table 2129. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_LENGTH fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY

*Table 2130. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_READONLY fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED

*Table 2131. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_UNSUPPORTED fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW

*Table 2132. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_RX_FIFO_OVERFLOW fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

#### i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW

*Table 2133. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TARGET_ERR_CNT_RI_INDIRECT_FIFO_OVERFLOW fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `CNT` | RW | 0x0 | Error count (saturates at 255). Write 0 to clear. |

### i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT

*Table 2134. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RX_DESC_QUEUE_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `RX_DESC` | R | 0x0 | RX Data |

### i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT

*Table 2135. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.RX_DATA_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `RX_DATA` | R | 0x0 | RX Data |

### i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT

*Table 2136. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TX_DESC_QUEUE_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `TX_DESC` | W | 0x0 | TX Data |

### i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT

*Table 2137. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.TX_DATA_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `TX_DATA` | W | 0x0 | TX Data |

### i3c_csr[6].I3C_EC.TTI.IBI_PORT

*Table 2138. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_PORT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `IBI_DATA` | W | 0x0 | IBI Data |

### i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE

*Table 2139. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `TX_DATA_BUFFER_SIZE` | R | 0x5 | Transmit Data Buffer Size in DWORDs calculated as `2^(N+1)` |
| 23:16 | `RX_DATA_BUFFER_SIZE` | R | 0x5 | Receive Data Buffer Size in DWORDs calculated as `2^(N+1)` |
| 15:8 | `TX_DESC_BUFFER_SIZE` | R | 0x4 | TX Descriptor Buffer Size in DWORDs calculated as `2^(N+1)` |
| 7:0 | `RX_DESC_BUFFER_SIZE` | R | 0x4 | RX Descriptor Buffer Size in DWORDs calculated as `2^(N+1)` |

### i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE

*Table 2140. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.IBI_QUEUE_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `IBI_QUEUE_SIZE` | R | 0x4 | IBI Queue Size in DWORDs calculated as `2^(N+1)` |

### i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL

*Table 2141. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.QUEUE_THLD_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `IBI_THLD` | RW | 0x1 | Controls the minimum number of IBI Queue entries needed to trigger the IBI threshold interrupt. |
| 23:16 | `Reserved` | - | - | Reserved |
| 15:8 | `RX_DESC_THLD` | RW | 0x1 | Controls the minimum number of TTI RX Descriptor Queue entries needed to trigger the TTI RX Descriptor interrupt. |
| 7:0 | `TX_DESC_THLD` | RW | 0x1 | Controls the minimum number of empty TTI TX Descriptor Queue entries needed to trigger the TTI TX Descriptor interrupt. |

### i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL

*Table 2142. oca_i3c_wrap.i3c_csr[6].I3C_EC.TTI.DATA_BUFFER_THLD_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:27 | `Reserved` | - | - | Reserved |
| 26:24 | `RX_START_THLD` | RW | 0x1 | Minimum number of TTI RX Data queue entries of data received, in DWORDs, that will trigger the TTI RX Data interrupt. Interrupt triggers when `2^(N+1)` RX Buffer DWORD entries are received during the Read transfer. |
| 23:19 | `Reserved` | - | - | Reserved |
| 18:16 | `TX_START_THLD` | RW | 0x1 | Minimum number of available TTI TX Data queue entries, in DWORDs, that will trigger the TTI TX Data interrupt. Interrupt triggers when `2^(N+1)` TX Buffer DWORD entries are available. |
| 15:11 | `Reserved` | - | - | Reserved |
| 10:8 | `RX_DATA_THLD` | RW | 0x1 | Minimum number of TTI RX Data queue entries of data received, in DWORDs, that will trigger the TTI RX Data interrupt. Interrupt triggers when `2^(N+1)` RX Buffer DWORD entries are received during the Read transfer. |
| 7:3 | `Reserved` | - | - | Reserved |
| 2:0 | `TX_DATA_THLD` | RW | 0x1 | Minimum number of available TTI TX Data queue entries, in DWORDs, that will trigger the TTI TX Data interrupt. Interrupt triggers when `2^(N+1)` TX Buffer DWORD entries are available. |

### i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER

*Table 2143. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:8 | `CAP_LENGTH` | R | 0x1A | Capability Structure Length in DWORDs |
| 7:0 | `CAP_ID` | R | 0xC1 | Extended Capability ID |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL

*Table 2144. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `PLACEHOLDER` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS

*Table 2145. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `PLACEHOLDER` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG

*Table 2146. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_CFG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Reserved` | - | - | Reserved |
| 1 | `REC_PAYLOAD_DONE` | RW | 0x0 | Inform Recovery Handler that payload transfer is finished. |
| 0 | `REC_INTF_BYPASS` | RW | 0x0 | Choose Recovery Interface access type:<br>• 0 - I3C Core<br>• 1 - direct AXI |

### i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS

*Table 2147. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.REC_INTF_REG_W1C_ACCESS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:16 | `INDIRECT_FIFO_CTRL_RESET` | RW | 0x0 | Write to 'Indirect memory configuration - reset' register, mocking a hardware access (bypassing 'write 1 to clear' register property). |
| 15:8 | `RECOVERY_CTRL_ACTIVATE_REC_IMG` | RW | 0x0 | Write to 'Recovery Control - Activate Recovery Image' register, mocking a hardware access (bypassing 'write 1 to clear' register property). |
| 7:0 | `DEVICE_RESET_CTRL` | RW | 0x0 | Write to 'Reset control - Device Reset Control' register, mocking a hardware access (bypassing 'write 1 to clear' register property). |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2

*Table 2148. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `PLACEHOLDER` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3

*Table 2149. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_RSVD_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `PLACEHOLDER` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF

*Table 2150. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_CONF fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | PAD_TYPE | RW | 0x1 | Select pad type<br><br>0 - Bidirectional<br><br>1 - Open-drain<br><br>2 - Input-only<br><br>3 - Analog input |
| 23:8 | Reserved | - | - | Reserved |
| 7 | VIRTUAL_OD_EN | RW | 0x0 | Enable virtual open drain:<br><br>0 - disabled<br><br>1 - enabled |
| 6 | OD_EN | RW | 0x0 | Enable Open-Drain:<br><br>0 - disabled<br><br>1 - enabled |
| 5 | IO_INVERSION | RW | 0x0 | Invert I/O signal:<br><br>0 - signals pass-through<br><br>1 - signals are inverted |
| 4 | PULL_EN | RW | 0x0 | Enable Pull:<br><br>0 - disabled<br><br>1 - enabled |
| 3 | PULL_DIR | RW | 0x0 | Direction of the pull:<br><br>0 - Pull down<br><br>1 - Pull up |
| 2 | KEEPER_EN | RW | 0x0 | Enable the High-Keeper:<br><br>0 - disabled<br><br>1 - enabled |
| 1 | SCHMITT_EN | RW | 0x0 | Enable the Schmitt Trigger:<br><br>0 - disabled<br><br>1 - enabled |
| 0 | INPUT_ENABLE | RW | 0x1 | Enable input:<br><br>0 - enabled<br><br>1 - disabled |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR

*Table 2151. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_PAD_ATTR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | DRIVE_STRENGTH | RW | 0xF | Select driver strength<br><br>'0 - lowest<br><br>'1 - highest |
| 23:16 | Reserved | - | - | Reserved |
| 15:8 | DRIVE_SLEW_RATE | RW | 0xF | Select driver slew rate<br><br>'0 - lowest<br><br>'1 - highest |
| 7:0 | Reserved | - | - | Reserved |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2

*Table 2152. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PLACEHOLDER | RW | 0x0 | Reserved for: I/O ring and pad configuration |

### i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3

*Table 2153. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.SOC_MGMT_FEATURE_3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | PLACEHOLDER | RW | 0x0 | Reserved for: I/O ring and pad configuration |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG

*Table 2154. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_R_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19:0 | T_R | RW | 0x0 | Rise time of both SDA and SCL in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG

*Table 2155. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_F_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19:0 | T_F | RW | 0x0 | Fall time of both SDA and SCL in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG

*Table 2156. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19:0 | T_SU_DAT | RW | 0x0 | Data setup time in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG

*Table 2157. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_DAT_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_SU_DAT_I2C` | RW | 0x0 | Data setup time in I2C mode in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG

*Table 2158. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_DAT_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HD_DAT` | RW | 0x0 | Data hold time in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG

*Table 2159. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HIGH` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG

*Table 2160. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_OD_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HIGH_OD` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG

*Table 2161. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_INIT_OD_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HIGH_INIT_OD` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG

*Table 2162. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HIGH_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HIGH_I2C` | RW | 0x0 | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG

*Table 2163. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_LOW` | RW | 0x0 | Low period of the SCL in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG

*Table 2164. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_OD_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_LOW_OD` | RW | 0x0 | Low period of the SCL in Open-Drain mode in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG

*Table 2165. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_LOW_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_LOW_I2C` | RW | 0x0 | Low period of the SCL in I2C mode in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG

*Table 2166. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HD_STA` | RW | 0x0 | Hold time for START in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG

*Table 2167. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_STA_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HD_STA_I2C` | RW | 0x0 | Hold time for START in I2C mode in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG

*Table 2168. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HD_RSTA_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HD_RSTA` | RW | 0x0 | Hold time for repeated START in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG

*Table 2169. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_SU_STA` | RW | 0x0 | Setup time for repeated START in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG

*Table 2170. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STA_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_SU_STA_I2C` | RW | 0x0 | Setup time for repeated START in I2C mode in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG

*Table 2171. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_SU_STO` | RW | 0x0 | Setup time for STOP in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG

*Table 2172. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_SU_STO_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_SU_STO_I2C` | RW | 0x0 | Setup time for STOP in in I2C mode clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG

*Table 2173. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_DS_OD_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_DS_OD` | RW | 0x0 | Setup time for SDA during START in clock units |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG

*Table 2174. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `T_FREE` | RW | 0xC | Bus Free Time Between a STOP and a START |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG

*Table 2175. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_FREE_I2C_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `T_FREE_I2C` | RW | 0xC | Bus Free Time Between a STOP and a START in I2C mode |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG

*Table 2176. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_AVAL_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `T_AVAL` | RW | 0x12C | - |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG

*Table 2177. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_IDLE_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `T_IDLE` | RW | 0xEA60 | Time in clock cycles from STOP detection until Bus Idle Condition (tIDLE). Configure based on System Clock |

### i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG

*Table 2178. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.HDR_TIMEOUT_EN_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `HDR_TIMEOUT_EN` | RW | 0x0 | Enable bit for the optional 60us HDR error recovery timer (I3C spec 5.1.10.1.9). When enabled, the Target can recover from TE0/TE1 errors if both SCL and SDA stay High for a period exceeding the configured threshold. Disabled on reset. |

### i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG

*Table 2179. oca_i3c_wrap.i3c_csr[6].I3C_EC.SoCMgmtIf.T_HDR_TIMEOUT_REG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `T_HDR_TIMEOUT` | RW | 0xEA60 | Timer threshold in clock cycles for the optional HDR error recovery (I3C spec 5.1.10.1.9). If both SCL and SDA stay High for this many clock cycles during a TE0/TE1 error-induced HDR mode, the Target exits HDR mode and returns to Idle. Configure based on System Clock to achieve 60us. |

### i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER

*Table 2180. oca_i3c_wrap.i3c_csr[6].I3C_EC.CtrlCfg.EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:8 | `CAP_LENGTH` | R | 0x2 | Capability Structure Length in DWORDs |
| 7:0 | `CAP_ID` | R | 0x2 | Extended Capability ID |

### i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG

*Table 2181. oca_i3c_wrap.i3c_csr[6].I3C_EC.CtrlCfg.CONTROLLER_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5:4 | `OPERATION_MODE` | R | 0x1 | - |
| 3:0 | `Reserved` | - | - | Reserved |

### i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER

*Table 2182. oca_i3c_wrap.i3c_csr[6].I3C_EC.TERMINATION_EXTCAP_HEADER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 23:8 | CAP_LENGTH | R | 0x1 | Capability Structure Length in DWORDs |
| 7:0 | CAP_ID | R | 0x0 | Extended Capability ID |

### i3c_csr[6].DAT.DAT_MEMORY[128]

*Table 2183. oca_i3c_wrap.i3c_csr[6].DAT.DAT_MEMORY[128] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:59 | Reserved | - | - | Reserved |
| 58:51 | AUTOCMD_HDR_CODE | RW | - | Device auto command in HDR mode |
| 50:48 | AUTOCMD_MODE | RW | - | Auto command mode and speed |
| 47:40 | AUTOCMD_VALUE | RW | - | IBI value that triggers auto command |
| 39:32 | AUTOCMD_MASK | RW | - | IBI mask |
| 31 | DEVICE | RW | - | Device type: 0 - I3C device, 1 - I2C device. |
| 30:29 | DEV_NACK_RETRY_CNT | RW | - | Number of retries before giving up |
| 28:26 | RING_ID | RW | - | Send IBI read to ring bundle |
| 25:24 | Reserved | - | - | Reserved |
| 23:16 | DYNAMIC_ADDRESS | RW | - | I3C dynamic address |
| 15 | TS | RW | - | Enable/disable IBI timestamp |
| 14 | CRR_REJECT | RW | - | Reject device's request for controller change |
| 13 | IBI_REJECT | RW | - | Reject device's request for IBI |
| 12 | IBI_PAYLOAD | RW | - | Device's IBI contains data payload |
| 11:7 | Reserved | - | - | Reserved |
| 6:0 | STATIC_ADDRESS | RW | - | I3C/I2C static device address |

### i3c_csr[6].DCT.DCT_MEMORY[128]

*Table 2184. oca_i3c_wrap.i3c_csr[6].DCT.DCT_MEMORY[128] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 127:104 | Reserved | - | - | Reserved |
| 103:96 | DYNAMIC_ADDRESS | R | - | Device I3C Dynamic Address after ENTDAA |
| 95:80 | Reserved | - | - | Reserved |
| 79:72 | BCR | R | - | Value of the I3C device's Bus Characteristics Register |
| 71:64 | DCR | R | - | Value of the I3C device's Device Characteristics Register |
| 63:48 | Reserved | - | - | Reserved |
| 47:32 | PID_LO | R | - | Device Provisional ID Low |
| 31:0 | PID_HI | R | - | Device Provisional ID High |

## AVSBus Controller

### Overview

The AVSBus Controller is a hardware IP that implements the AVSBus 1.3.1 specification for adaptive voltage scaling communication. It acts as a controller (master) that communicates with a single AVS target device (slave) to manage voltage rails, monitor power conditions, and perform real-time voltage adjustments for optimal performance and

energy efficiency. This version supports 32-bit frame-based communication without multi-target addressing.

### Description and Features

- **Single-Target Support**: Communicates with a single AVS target device through a two-wire protocol interface
- **AXI4-Lite Register Interface**: A 32-bit AXI4-Lite port, bridged internally to the APB4 register file
- **Command FIFO**: 8-deep command queue, crossing from the register clock domain to the AVS clock domain
- **Readback FIFO**: 8-deep response buffer with automatic hardware retry on a failed acknowledgment or a bad CRC
- **Flexible Clock Configuration**: AVS clock generated from either the register clock or a reference clock, with a programmable divider and duty cycle
- **Maskable Interrupts**: Nine sources covering FIFO overruns, retry exhaustion, an unresponsive target and target-signalled interrupts
- **Slave Resync Operation**: Automatic periodic resynchronization at a configurable interval, or forced by software
- **CRC-3 Protection**: Hardware generation on every command and checking on every response

### Architecture

The AVSBus Controller implements an Adaptive Voltage Scaling Bus (AVSBus) 1.3.1 controller for dynamic voltage and frequency scaling (DVFS) in single-target systems, communicating with one AVS target device over a two-wire protocol interface.

The design is a register file, a pair of asynchronous FIFOs that carry commands and responses between the register and AVS clock domains, a protocol FSM that serializes and deserializes subframes, a CRC-3 engine shared between transmit and receive, and the AVS clock generation logic.

Figure 72 shows the FIFO, protocol-engine, and clock-generation paths. The two clock-domain boxes hold the logic clocked by `clk_reg_i` and by the generated `avs_clk`; the FIFOs and synchronizers that straddle the gap between them carry every crossing. The clock generation is drawn in full in Figure 74, and the protocol FSM in Figure 73.

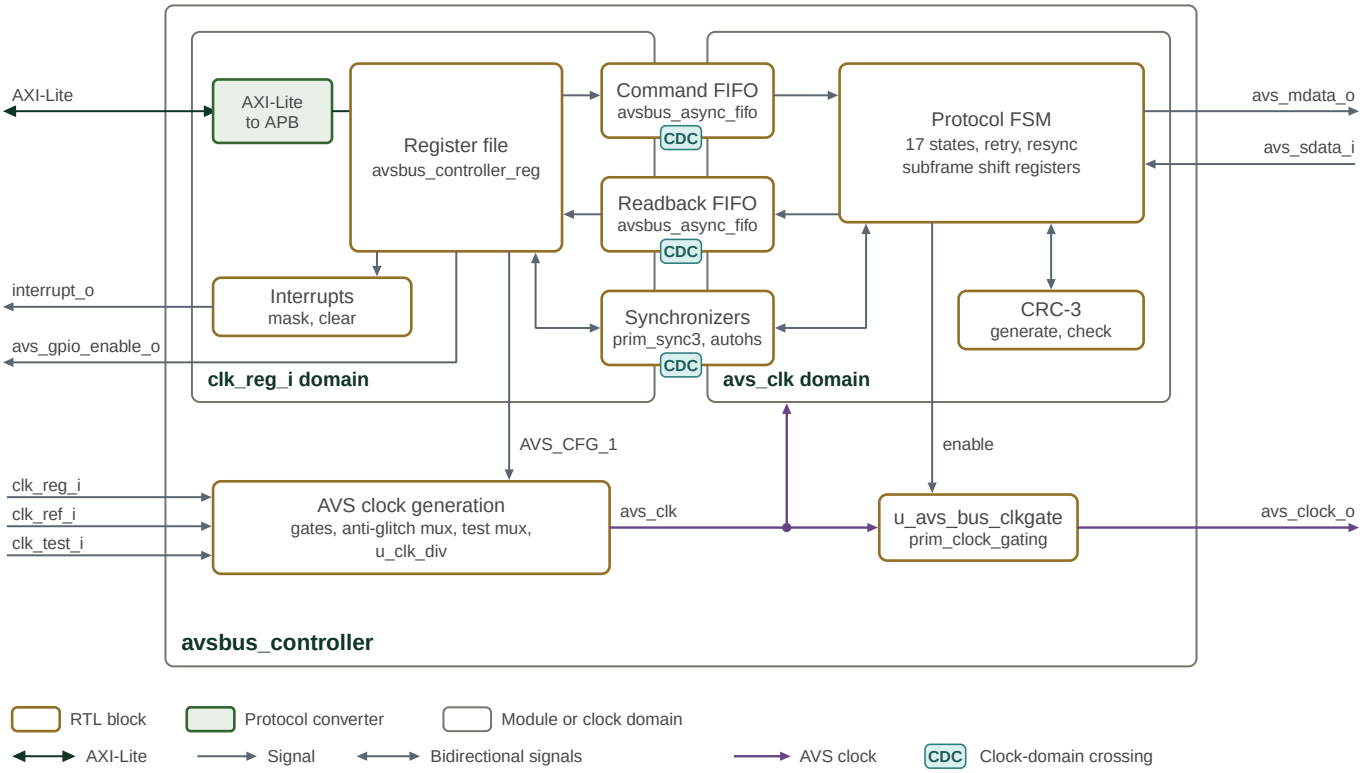


*Figure 72. AVSBus controller block diagram*

### AVS Protocol Engine

Software queues commands by writing `AVS_CMD`, which pushes a 32-bit word into the command FIFO. The protocol FSM pops each command, prepends the fixed preamble, appends a calculated CRC-3, and shifts the resulting 32-bit subframe out on `avs_mdata_o`. The target's reply arrives one subframe later: it is shifted in from `avs_sdata_i` during the 32 cycles that carry the next command, checked for a valid frame bit and a good CRC, and pushed into the readback FIFO.

Retries are automatic. A slave acknowledgment of 1 (resource unavailable) or 2 (bad CRC) causes the master to resend the command, up to `AVS_CFG_0.MAX_RETRIES` times after the original attempt. If the last retry also fails, the failing response is delivered to software and `MAX_RETRIES_ATTEMPTED_INT` is raised. Setting `MAX_RETRIES` to 0 suppresses both the retries and that interrupt; the failing response is delivered to software instead. A reply whose frame valid bit is set means the target did not respond at all and raises `SLAVE_UNRESPONSIVE_INT`. There is no separate programmable response timeout.

The master also issues periodic slave resync operations. The FSM stays in `AVS_SLAVE_RESYNC` for 35 `avs_clk` cycles, while its resync counter runs from 0 to 34, and holds `avs_mdata_o` high throughout. `avs_mdata_o` is also high in the cycles around that state, through idle or the receive half of a retry beforehand and through the launch cycle afterwards, so the high period on the pin is longer than the resync state itself.

`AVS_CFG_0.RESYNC_INTERVAL` is the number of `clk_reg_i` cycles the register domain counts, from when it observes the end of the previous resync, before it requests the next one. `AVS_CFG_1.FORCE_SLAVE_RESYNC_OPERATION` requests one immediately. Either request is registered in the `clk_reg_i` domain and crosses into the AVS domain through a three-stage `avs_clk` synchronizer. The master stops launching new commands once it sees the request, and starts the resync from idle after the frame in flight and its reply complete.

### Clock Generation and Management

Table 2185 identifies the register, reference, and generated AVS clocks.

*Table 2185. AVSBus clock sources*

| Clock Source | Description |
|---|---|
| clk_reg_i | Register interface clock, synchronous with register access |
| clk_ref_i | Reference clock, independent of AVS bus timing |
| Selection | Software-configurable via `AVS_CFG_1.AVS_CLOCK_SELECT` |

The selected source passes through an anti-glitch mux and then a programmable divider, whose ratio and duty cycle are both 8-bit values in `AVS_CFG_1`. Out of reset the controller runs on `clk_ref_i` divided by 4 at a 50% duty cycle. Both divider fields reset to 0, which is a sentinel: the divider keeps its hardware defaults until software writes a non-zero value.

Changing the clock configuration is not glitch-free on its own. Set `AVS_CFG_1.TURN_OFF_ALL_PREMUX_CLOCKS` to gate both source clocks before changing the source selection or the divider settings, then clear it afterwards.

Power is saved at two levels: `AVS_CFG_1.STOP_AVS_CLOCK_ON_IDLE` gates the AVS clock whenever the bus is idle, and `TURN_OFF_ALL_PREMUX_CLOCKS` gates the clocks entering the mux. Restarting a gated AVS clock always begins with a slave resync.

### CRC Verification Engine

The `avsbus_crc3` module computes a 3-bit CRC combinationally over a 32-bit subframe. Two instances are used. The generate instance is fed the outgoing master subframe with its CRC field zeroed, and its output becomes the three bits shifted out last. The check instance is fed the received slave subframe intact, including the CRC the target sent, and reports a good check when the result is zero. A failed check prevents the subframe from updating `AVS_LATEST_SLAVE_SUBFRAME` and `AVS_SLAVE_STATUS`, and the retry logic resends the command.

### Command and Response FIFO System

Table 2186 compares the command and readback FIFO roles and operation.

*Table 2186. AVSBus command and readback FIFOs*

| FIFO | Command FIFO | Readback FIFO |
|---|---|---|
| Depth | `COMMAND_FIFO_DEPTH`, 8 entries by default | `READBACK_FIFO_DEPTH`, 8 entries by default |
| Direction | Register clock domain to AVS clock domain | AVS clock domain to register clock domain |
| Occupancy | `AVS_FIFOS_STATUS.CMD_FIFO_*` | `AVS_FIFOS_STATUS.READBACK_FIFO_*` |
| Overrun behaviour | A write to `AVS_CMD` when full is dropped, completes with an error response and raises `CMD_FIFO_OVERFLOW_INT` | A response received when full is dropped and raises `READBACK_OVERFLOW_INT` |

Both are instances of `avsbus_async_fifo`, which requires a power-of-two depth. Pointers cross between the two clock domains as Gray codes through three-stage synchronizers.

While the readback FIFO is full the master launches no further commands, but replies to commands already launched are still pushed as they arrive. A reply that meets a full FIFO is dropped and raises `READBACK_OVERFLOW_INT`, so a software reader that stops draining `AVS_READBACK` stalls new commands and can lose the responses to commands already on the bus.

### Reset and Initialization

The controller takes three resets. `rst_clk_div_ni` resets the programmable clock divider, `rst_reg_ni` resets the register file and the protocol logic, and `rst_ref_ni` resets the `clk_ref_i` leg of the anti-glitch clock mux.

**`rst_clk_div_ni` must be deasserted before `rst_reg_ni`**. The divider has to be producing a clock before the logic it feeds leaves reset, otherwise the AVS clock domain never starts.

Because the divider defaults to divide-by-4 on `clk_ref_i` and the protocol FSM starts as soon as reset is released, the controller begins issuing slave resync operations without any software configuration. Configuration is therefore a matter of adjusting defaults rather than enabling the block. The AVSBus configuration sequence is in the Programmer's Guide.

### Data Flow and Control Path

#### Command Path

1. Software writes `AVS_CMD`; the 32-bit word enters the command FIFO. If the FIFO is full, the write is dropped and completes with an error response.
2. The protocol FSM pops a command, builds the master subframe as preamble, `AVS_CMD[29:3]` and CRC-3, and shifts it out on `avs_mdata_o`.
3. The target's reply shifts in on `avs_sdata_i` during the following subframe.
4. A reply that is valid and passes CRC updates `AVS_SLAVE_STATUS` and `AVS_LATEST_SLAVE_SUBFRAME` and is pushed into the readback FIFO; otherwise the retry logic resends the command.
5. Software reads `AVS_READBACK`, which pops the readback FIFO.

A dropped write also raises `CMD_FIFO_OVERFLOW_INT`. `AVS_NORMAL_STATUS.CMD_FIFO_FULL` and `AVS_FIFOS_STATUS.CMD_FIFO_VACANT_SLOTS` report the command FIFO's space.

#### Response Inspection

`AVS_READBACK` is the normal path and advances the FIFO on every read that finds it non-empty; a read of an empty FIFO returns 0 with an error response. `AVS_DEBUG_READBACK` mirrors the top of the FIFO without advancing it, and reads 0xFFFFFFFF when the FIFO is empty. `AVS_LATEST_SLAVE_SUBFRAME` holds the most recent valid subframe regardless of FIFO state, which is what to read when the question is what the target is doing now rather than what it said several commands ago.

### Clock Domain Architecture

Two domains carry functional logic. The register domain, clocked by `clk_reg_i`, holds the AXI4-Lite bridge, the register file, the command FIFO write side, the readback FIFO read side and interrupt generation. The AVS domain, clocked by the generated `avs_clk`, holds the protocol FSM, the shift registers and the CRC engines. `avs_clock_o` is `avs_clk` after the idle clock gate, so the FSM keeps running while `STOP_AVS_CLOCK_ON_IDLE` holds the pin clock stopped.

FIFO pointers cross between the domains as Gray codes. Individual control and status bits cross through `prim_sync3` three-stage synchronizers, and pulses through `prim_sync3_pulse`. The reset synchronizers are six stages deep and take their scan bypass from `scan_rst_ni`.

### Protocol State Machine

`cur_state_debug_o` exposes the FSM state as a 17-bit one-hot vector, resynchronized to `clk_reg_i`. The states, in bit order

from 0, are (Table 2187).

*Table 2187. AVSBus protocol state encodings*

| Bit | State | Meaning |
|---|---|---|
| 0 | AVS_RESET | Out of reset, before the first resync |
| 1 | AVS_SLAVE_RESYNC | Holding `avs_mdata_o` high for 35 cycles while the resync counter runs from 0 to 34 |
| 2 | AVS_LAUNCH_FRAME_POST_RESYNC | First command launch after a resync |
| 3 | AVS_IDLE | No command queued; the AVS clock may be gated |
| 4, 5 | AVS_SHIFT_1ST_SUBFRAME, AVS_END_1ST_SUBFRAME | First subframe after idle or resync, launching a command |
| 6, 7 | AVS_SHIFT_MID_SUBFRAME, AVS_END_MID_SUBFRAME | Transmitting one command while receiving the reply to the previous one |
| 8, 9 | AVS_SHIFT_LAST_SUBFRAME, AVS_END_LAST_SUBFRAME | Final subframe, draining the last reply with no new command to send |
| 10-15 | AVS_RETRY_* | The same shift and end pairs used while retrying a failed command, split into transmit-and-receive, transmit-only and receive-only variants |
| 16 | AVS_PROCESS_PREVIOUS_SDATA | Evaluating a received subframe |

Command transmission and response reception overlap: the reply to a command arrives during the subframe that carries the next command, which is why the mid-subframe states do both at once.

Figure 73 shows the transitions between these states, grouping the 17 states into 11 boxes. Each subframe box stands for a SHIFT state, which holds for the 32 bits of a subframe, and the END state that follows it, where the transition is decided; the three retry boxes pair the six `AVS_RETRY_*` states the same way. The figure omits the self-loops of states that wait: idle, the resync and each SHIFT state. Resync done means that the resync counter has reached 34. RDY means that a command is waiting, the readback FIFO has room and no resync is due. Stop on idle is `AVS_CFG_1.STOP_AVS_CLOCK_ON_IDLE`: with it set, a command waiting in idle restarts the AVS clock through a resync instead of launching directly. A retry transition is taken when the received subframe has a bad CRC, a resource-unavailable or bad-CRC acknowledgment, or no response, and `MAX_RETRIES` is above zero; with `MAX_RETRIES` at zero the RDY and not-RDY transitions apply instead. A reply that arrives during a retry of the preceding command is held and processed after the retries finish. Every retry sequence ends with a slave resync.

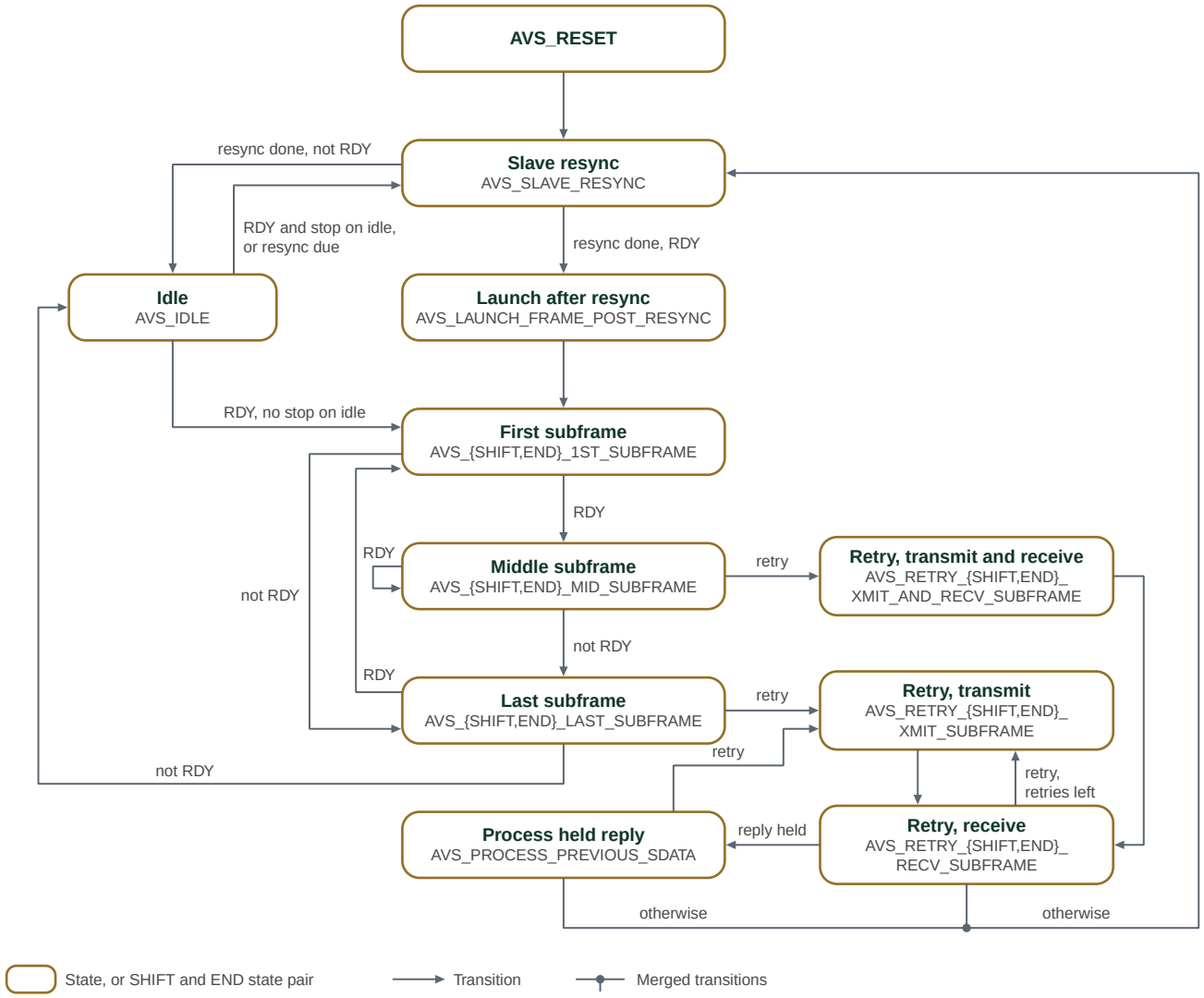


*Figure 73. AVSBus protocol state machine*

### Error Handling and Diagnostics

Table 2188 describes the controller response to each error condition.

*Table 2188. AVSBus error handling*

| Condition | Handling |
|---|---|
| Bad CRC from the target | Automatic retry, up to `AVS_CFG_0.MAX_RETRIES` |
| Resource unavailable acknowledgment | Automatic retry, up to `AVS_CFG_0.MAX_RETRIES` |
| Bad data, data type or selector acknowledgment | No retry; the response is delivered to software |
| Retry budget exhausted | `MAX_RETRIES_ATTEMPTED_INT` once the last of `MAX_RETRIES` retries also fails, and the failing response is delivered to software; not raised while `MAX_RETRIES` is 0. `AVS_NORMAL_STATUS.TOTAL_RETRIES` counts every retry attempted |
| Target not responding | `SLAVE_UNRESPONSIVE_INT` |
| Command FIFO overrun | Write dropped with an error response, `CMD_FIFO_OVERFLOW_INT` |
| Readback FIFO overrun | Response dropped, `READBACK_OVERFLOW_INT` |
| Read of an empty readback FIFO | Returns 0 with an error response, `READBACK_UNDERFLOW_INT` |

For diagnosis, AVS_NORMAL_STATUS reports bus idle, resync, retry-in-progress and FIFO states; AVS_FIFOS_STATUS reports occupied and vacant slots in both FIFOs; AVS_SLAVE_STATUS and AVS_LATEST_SLAVE_SUBFRAME report the target's most recent acknowledgment and status response; and cur_state_debug_o exposes the FSM state to the surrounding system.

### Interface

The AVSBus Controller presents an AXI4-Lite register interface, a two-wire AVSBus protocol interface, and clock, reset, interrupt and DFT interfaces for system integration. The register file itself is APB4; the controller contains an axi_lite_to_apb bridge, so APB4 is not exposed at the module boundary.

The parameters and ports of avsbus_controller are listed in the HTML edition of this manual, under RTL Modules Reference.

### AVSBus Frame Format

Both subframes are 32 bits: 29 bits of payload followed by a 3-bit CRC. The master subframe carries a fixed 2'b01 preamble in its two most significant bits; the remaining fields occupy the same bit positions as the AVS_CMD register, so software writes a command in wire order.

Table 2189 defines the fields transmitted in a master subframe.

*Table 2189. AVSBus Master Subframe*

| Field Name | Bit Range | Width | Description |
|---|---|---|---|
| Preamble | [31:30] | 2 | Fixed 2'b01, inserted by hardware |
| Command Type | [29:28] | 2 | AVS_CMD.R_OR_W: 0 = commit write, 1 = hold write, 2 = reserved, 3 = read |
| Command Group | [27] | 1 | AVS_CMD.CMD_GRP: 0 = standard, 1 = manufacturer-specific |
| Command Code | [26:23] | 4 | AVS_CMD.CMD_CODE, see Table 2190 |
| Rail Select | [22:19] | 4 | AVS_CMD.RAIL_SEL: rail index, 0xF = broadcast |
| Command Data | [18:3] | 16 | AVS_CMD.CMD_DATA; 0xFFFF for read commands |
| CRC-3 | [2:0] | 3 | Calculated by hardware over bits [31:3] |

Table 2190 lists the command-group-zero data types, resolutions, and access modes.

*Table 2190. AVSBus Command Codes (Command Group 0)*

| Code | Data Type | Resolution | Description |
|---|---|---|---|
| 0x0 | Target Rail Voltage | 1 mV | 16-bit unsigned, read or write |
| 0x1 | Vout Transition Rate | 1 mV/us | MSByte rise rate, LSByte fall rate |
| 0x2 | Rail Current | 10 mA | 16-bit unsigned, read only |
| 0x3 | Temperature | 0.1 degC | 16-bit signed, read only |
| 0x4 | Force Voltage Reset | N/A | Requires a command data field of 0x0000 |
| 0x5 | Power Mode | N/A | 3 bits, LSB aligned: 0x0 maximum efficiency, 0x3 maximum power, 0x4-0x7 manufacturer-specific |
| 0xE | AVSBus Status | N/A | 16 bits, encoded per the AVS specification |
| 0xF | AVSBus Version | N/A | 4 bits, LSB aligned, read only |

With command group 1 the command code is manufacturer-specific and Table 2191 above does not apply.

*Table 2191. AVSBus Slave Subframe*

| Field Name | Bit Range | Width | Description |
|---|---|---|---|
| Slave Acknowledgment | [31:30] | 2 | 0 = action performed, 1 = good CRC but resource unavailable, 2 = bad CRC received, 3 = good CRC but bad data, data type or selector. Codes 1 and 2 trigger a hardware retry |
| Frame Valid | [29] | 1 | 0 in a valid frame. A 1 means the target did not respond and raises `SLAVE_UNRESPONSIVE_INT` |
| Status Response | [28:24] | 5 | Per the AVS specification: bit 4 VDone, bit 3 StatusAlert, bit 2 AVS_Control, bits 1-0 manufacturer-specific |
| Response Data | [23:8] | 16 | Read data; 0xFFFF when the master issued a write |
| Reserved | [7:3] | 5 | Reserved |
| CRC-3 | [2:0] | 3 | CRC received from the target, checked by hardware |

A received subframe reaches software by two paths. It is pushed into the readback FIFO and read out of `AVS_READBACK`, which pops the FIFO, or inspected without popping through `AVS_DEBUG_READBACK`. Independently of the FIFO, a subframe whose frame valid bit is 0 and whose CRC checks also latches into `AVS_LATEST_SLAVE_SUBFRAME` and `AVS_SLAVE_STATUS`, so the current target status is visible without waiting for the FIFO to drain.

### Clock Architecture

The AVS clock is derived from one of two sources, optionally divided, and optionally gated while the bus is idle.

`AVS_CFG_1.AVS_CLOCK_SELECT[1:0]` selects both the source and whether the divider is used: bit 1 picks the source (0 = `clk_reg_i`, 1 = `clk_ref_i`) and bit 0 picks the divider (0 = bypass, 1 = divided). The reset value is `2'b11`, so the controller comes up on a divided `clk_ref_i`.

Figure 74 shows the clock path. Each source passes a clock gate that `TURN_OFF_ALL_PREMUX_CLOCKS` closes, then the anti-glitch mux `u_refclk_apbclk_mux` selects one of them, choosing `clk_ref_i` while in reset. The test mux substitutes `clk_test_i` in test mode. Inside `u_clk_div`, the glitch-free post-divider mux selects either the undivided clock or the divider output. The result, `avs_clk`, clocks the AVS domain and reaches `avs_clock_o` through `u_avs_bus_clkgate`, which the protocol FSM closes while the bus is idle if `STOP_AVS_CLOCK_ON_IDLE` is set. Each mux passes its input 0 when its select is 0. Table 2192 lists the four settings.

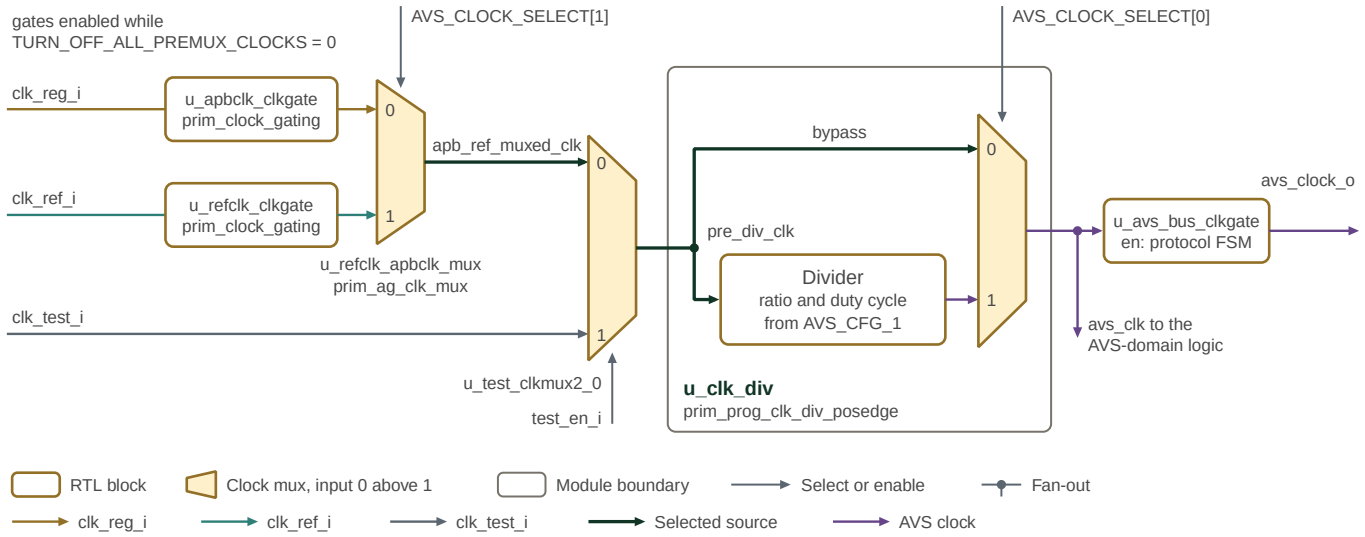


*Figure 74. AVSBus clock selection*

*Table 2192. AVS_CLOCK_SELECT settings*

| AVS_CLOCK_SELECT | Source mux | Post-divider mux | AVS clock |
|---|---|---|---|
| 2'b00 | 0 | 0, bypass | `clk_reg_i` |
| 2'b01 | 0 | 1, divided | `clk_reg_i` divided by `CLK_DIVIDER_VALUE` |
| 2'b10 | 1 | 0, bypass | `clk_ref_i` |
| 2'b11 (reset) | 1 | 1, divided | `clk_ref_i` divided by `CLK_DIVIDER_VALUE`, by 4 until software writes a non-zero value |

Table 2193 defines the source-selection and divider configuration fields.

*Table 2193. AVS Clock Configuration Fields*

| Field | Reset | Description |
|---|---|---|
| AVS_CFG_1.AVS_CLOCK_SELECT[1:0] | 0x3 | Clock source and divider bypass |
| AVS_CFG_1.CLK_DIVIDER_VALUE[23:16] | 0x0 | Division ratio. A value below 2 is treated as 2. The reset value 0 is a sentinel that leaves the divider at its hardware default of 4 |
| AVS_CFG_1.CLK_DIVIDER_DUTY_CYCLE_NUMERATOR[31:24] | 0x0 | High-pulse fraction in 1/256 units, so 128 is 50%. A 50% duty cycle is only reachable with an even divider. The reset value 0 is a sentinel that leaves the numerator at its hardware default of 0x80 |
| AVS_CFG_1.STOP_AVS_CLOCK_ON_IDLE[8] | 0x0 | Gate the AVS clock while the bus is idle. The master issues a slave resync whenever the clock restarts |
| AVS_CFG_1.TURN_OFF_ALL_PREMUX_CLOCKS[10] | 0x0 | Gate both clocks feeding the source mux |

Gate the clocks with `TURN_OFF_ALL_PREMUX_CLOCKS` before changing `AVS_CLOCK_SELECT`, `CLK_DIVIDER_VALUE` or `CLK_DIVIDER_DUTY_CYCLE_NUMERATOR`, then release the gate afterwards.

While `test_en_i` is high the source and divider-bypass muxes ignore `AVS_CLOCK_SELECT`, and `clk_test_i` drives the AVS clock directly.

### Reset Architecture

Table 2194 identifies the logic reset by each AVSBus reset input.

*Table 2194. AVSBus Reset Domains*

| Reset | Scope |
|---|---|
| rst_clk_div_ni | The programmable clock divider |
| rst_reg_ni | The register file, the protocol FSM and the reset synchronizers for the register, pre-divider and AVS clock domains |
| rst_ref_ni | The `clk_ref_i` leg of the anti-glitch source mux |

`rst_clk_div_ni` must be deasserted before `rst_reg_ni`. The divider has to be producing a clock before the logic it feeds leaves reset, otherwise the AVS clock domain never starts.

### Interrupt Interface

`interrupt_o` is a level output, asserted while any `AVS_INTERRUPT` bit is set and its corresponding `AVS_INTERRUPT_MASK` bit is clear.

Table 2195 maps AVS_INTERRUPT bits to the events that set them.

*Table 2195. AVSBus Interrupt Sources*

| Bit | Field | Raised when |
|---|---|---|
| 8 | READBACK_OVERFLOW_INT | A target response was dropped because the readback FIFO was full |
| 7 | READBACK_UNDERFLOW_INT | `AVS_READBACK` was read while the readback FIFO was empty; the read returned 0 with an error response |
| 6 | CMD_FIFO_OVERFLOW_INT | `AVS_CMD` was written while the command FIFO was full, so the command was dropped and the write completed with an error response |
| 5 | MAX_RETRIES_ATTEMPTED_INT | A command still failed after `AVS_CFG_0.MAX_RETRIES` retries, `MAX_RETRIES` + 1 attempts in all. Never raised while `MAX_RETRIES` is 0 |
| 4 | SLAVE_UNRESPONSIVE_INT | The target did not drive a valid subframe |
| 3 | READBACK_HAS_DATA_INT | The readback FIFO holds data for software |
| 2 | READBACK_FIFO_FULL_INT | The readback FIFO is full; no further commands launch until it drains |
| 1 | CMD_FIFO_FULL_INT | The command FIFO is full |
| 0 | AVS_SLAVE_ISSUED_INTERRUPT | The target signalled an interrupt |

`AVS_INTERRUPT_MASK` is a **disable** mask and every bit resets to 1, so all sources are masked after reset and software must clear a bit to enable that source. Interrupt bits are cleared by writing 1 to the matching bit of `AVS_INTERRUPT_CLEAR`; those bits self-clear.

### Pad Enable Interface

`AVS_CONFIG.AVS_GPIO_ENABLE` drives `avs_gpio_enable_o`, which enables the pads carrying the AVSBus signals: output enable for the `avs_clock` and `avs_mdata` pins and input enable for the `avs_sdata` pin. It resets to 1, because the protocol FSM begins running as soon as reset is deasserted.

### Timing

Table 2196 summarises the subframe and clock timing.

*Table 2196. AVSBus Protocol Timing*

| Parameter | Value |
|---|---|
| Subframe length | 32 `avs_clock_o` cycles |
| Slave resync | 35 `avs_clk` cycles in `AVS_SLAVE_RESYNC` with `avs_mdata_o` held high; the pin stays high for longer, through the cycles around that state |
| Minimum AVS clock division | 2, when the divider is selected |
| Resync interval | `AVS_CFG_0.RESYNC_INTERVAL` `clk_reg_i` cycles, counted from when the register domain observes the end of the previous resync, plus one `clk_reg_i` cycle to register the request, three `avs_clk` cycles to synchronize it, and the time for any frame in flight to complete |

The resync counts are `avs_clk` cycles at the protocol FSM's state register. Pin-level setup, hold and propagation figures are a function of the implementation and are not specified by this IP.

### Integration Example

The register port takes the AXI4-Lite structs from `avsbus_controller_pkg`.

```
avsbus_controller #(
    .COMMAND_FIFO_DEPTH  (8),
    .READBACK_FIFO_DEPTH (8)
) u_avs_ctrl (
```

```
    // Clocks and resets
    .clk_reg_i      (reg_clk),
    .clk_ref_i      (ref_clk),
    .rst_ref_ni     (ref_rst_n),
    .rst_reg_ni     (reg_rst_n),
    .rst_clk_div_ni (clk_div_rst_n),  // deassert before rst_reg_ni

    // AVSBus interface
    .avs_sdata_i       (avs_sdata),
    .avs_mdata_o       (avs_mdata),
    .avs_clock_o       (avs_clock),
    .avs_gpio_enable_o (avs_pad_enable),

    // AXI4-Lite register interface
    .axil_req_i  (avs_axil_req),
    .axil_resp_o (avs_axil_resp),

    .interrupt_o (avs_irq),

    // DFT
    .scan_rst_ni (scan_rst_n),
    .test_en_i   (test_en),
    .clk_test_i  (test_clk),

    .cur_state_debug_o (avs_fsm_state)
);
```

### Memory Map and Register Reference

#### Memory Map

The AVSBus Controller occupies a 128-byte address space (0x00-0x7F) reached through its AXI4-Lite port. The register map holds the command and readback FIFO windows, status registers for monitoring the bus and the target, interrupt status, mask and clear registers, and configuration registers for the clock, the retry budget and the resync interval.

#### Address Alignment and Access Requirements

All registers are 32 bits wide and aligned to a 4-byte boundary. The highest allocated offset is `0x58`; offsets `0x5C` through `0x7F` are unallocated and reserved.

The controller decodes only the low 7 address bits, so the integrating address decoder is responsible for confining the block to its 128-byte window. The SMC memory map states how the SMC answers an access outside it.

Stored registers honour byte strobes, so a partial write updates only the addressed byte lanes. The two FIFO windows do not:

- `AVS_CMD` ignores write strobes and pushes the whole 32-bit write data word into the command FIFO, so it must be written with a single full-width access. A partial write enqueues a malformed command.
- `AVS_READBACK` pops the readback FIFO on any read, whatever the access width, so it must be read exactly once per

response. AVS_DEBUG_READBACK returns the same data without popping.

Neither FIFO window back-pressures the bus; both report a failed access instead. A write to AVS_CMD while the command FIFO is full is dropped, and a read of AVS_READBACK while the readback FIFO is empty returns 0 without popping. Each case raises its interrupt and completes the access with an error response, so software can detect the condition from the bus response alone. An all-zero word is also a well-formed target response, so the read data does not identify an empty read; only the error response or READBACK_UNDERFLOW_INT does. AVS_DEBUG_READBACK reads 0xFFFFFFFF when the FIFO is empty and never errors, which makes it the safer register to poll.

### Detailed Register Map

#### AVSBus Controller Address Map

Command, readback, status, and interrupt registers for the AVSBus (V1.3.1) controller.

*Table 2197. AVSBus Controller Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | AVS_CMD | W | The command+data to be transferred to the AVS bus |
| 0x4 | AVS_READBACK | R | Response data from AVS slave device (updated in response to both AVS read and write commands). Reading this register causes the AVS response readback fifo pointer to advance. |
| 0x8 | AVS_DEBUG_READBACK | R | This register mirrors the AVS_READBACK register, but reading it does NOT affect the AVS readback fifo pointer. Provided for debug purposes. |
| 0xC | AVS_LATEST_SLAVE_SUBFRAME | R | This register contains the most recently received subframe from the AVS slave, bypassing the AVS readback fifo. It is provided so that APB can see the current slave StatusResponse and SlaveAck settings in a timely fashion, since the status data from the next entry in the readback fifo might have considerable latency relative to when it was initially received from the AVS Slave. |
| 0x20 | AVS_NORMAL_STATUS | R | Various Status bits indicating normal operating conditions for APB2AVS bridge block. These all have corresponding maskable bits in the AVS_INTERRUPT register |
| 0x24 | AVS_SLAVE_STATUS | R | Contains the most recent status feedback from the AVS slave device |
| 0x28 | AVS_FIFOS_STATUS | R | Contains the number of occupied and vacant slots in the AVS master's command and readback FIFOs |
| 0x30 | AVS_INTERRUPT | R | Interrupt register - A '1' indicates an interrupt has occurred for that bit. To clear the interrupt bit, write to the corresponding field in the AVS_INTERRUPT_CLEAR register. |
| 0x34 | AVS_INTERRUPT_MASK | RW | Interrupt mask register - Determines which AVS_INTERRUPT bits can trigger an interrupt |
| 0x38 | AVS_INTERRUPT_CLEAR | W | Write 1 to clear corresponding interrupt bit |
| 0x50 | AVS_CFG_0 | RW | Configuration register for APB2AVS bridge block |
| 0x54 | AVS_CFG_1 | RW | Configuration register for APB2AVS bridge block |
| 0x58 | AVS_CONFIG | RW | AVS Configuration Register |

#### AVS_CMD

*Table 2198. avsbus_controller.AVS_CMD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:30 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 29:28 | R_OR_W | W | 0x0 | Read or write type: [list] **[]0x0 - Commit Write []**0x1 - Hold Write **[]0x2 - Reserved []**0x3 - Read [/list] |
| 27 | CMD_GRP | W | 0x0 | Command group: Set to 1 for manufacturer-specific Command Code (cmd_code), otherwise 0 |
| 26:23 | CMD_CODE | W | 0x0 | Command Code (For cmd_group=0. For cmd_group=1, this is manufacturer-specific): [list] **[]0x0 - Target Rail Voltage Read/Write []**0x1 - Vout Transition Rate Read/Write **[]0x2 - Rail Current Read []**0x3 - Temperature Read **[]0x4 - Force Voltage Reset (Requires wr_cmd_data=0x0) []**0x5 - Power Mode Read/Write **[]0xe - AVSBus Status Read/Write []**0xf - AVSBus Version Read [/list] |
| 22:19 | RAIL_SEL | W | 0x0 | Rail Select: 0x0 - Rail 0 0x1 - Rail 1 0xf - Broadcast |
| 18:3 | CMD_DATA | W | 0x0 | Write Cmd Data. Formats: [list] **[]Voltage (cmd_code 0x0): 16-bit unsigned int, 1LSB=1mV []**Transition Rate (cmd_code 0x1): [list] **[]MSByte: Rise Rate, 8-bit unsigned int, 1LSB=1mV/us []**LSByte: Fall Rate, 8-bit unsigned int, 1LSB=1mV/us [/list] **[]Reset Voltage (cmd_code 0x4): 16-bits, all zeroes []**Power Mode (cmd_code 0x5): 3-bits, LSB aligned: [list] **[]0x0 - Maximum Efficiency []**0x3 - Maximum Power **[]0x4-0x7 - Manufacturer-specific [/list] []**AVSBus Status (cmd_code 0xe): 16-bits, encoded per AVS Spec **[]AVSBus Version (cmd_code 0xf): 4 bits encoded per AVS Spec, LSB-aligned []**For read cmd_code's, this is 0xffff [/list] |
| 2:0 | Reserved | - | - | Reserved |

### AVS_READBACK

*Table 2199. avsbus_controller.AVS_READBACK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:30 | SLAVE_ACK | R RMOD | 0x0 | Indicates whether command executed or not - codes 0x1 and 0x2 trigger automatic retries by hardware: [list] **[]0x0: Action Performed []**0x1: Good CRC received but resource unavailable - hardware will retry **[]0x2: Bad CRC received - hardware will retry []**0x3: Good CRC received but bad data, data type or selector - no hardware retry [/list] |
| 29 | CONST0 | R RMOD | 0x0 | Always 0 |
| 28:24 | STATUS_RESPONSE | R RMOD | 0x0 | 5-bit field indicating general condition of the slave, per AVS spec StatusReponse field: [list] **[]bit 4: VDone - AND of all VDone bits of active rails []**bit 3: StatusAlert - Indicates potential status issue, requiring AVSBus Status read cmd for details **[]bit 2: AVS_Control - 1 if ABSBus is controlling at least 1 device output, 0 otherwise. []**bit 1: MfrSpcfc_Stts1 - Manufacturer-specific [*]bit 0: MfrSpcfc_Stts2 - Manufacturer-specific [/list] |
| 23:8 | CMD_DATA | R RMOD | 0x0 | Read Cmd Data. Format depends on cmd_code in previous master subframe:[list] **[]Voltage (cmd_code 0x0): 16-bit unsigned int, 1LSB=1mV []**Transition Rate (cmd_code 0x1): [list] **[]MSByte: Rise Rate, 8-bit unsigned int, 1LSB=1mV/us []**LSByte: Fall Rate, 8-bit unsigned int, 1LSB=1mV/us [/list] **[]Current (cmd_mode 0x2): 16-bit unsigned, 1LSB=10mA []**Temperature (cmd_code 0x3): 16-bit signed, 1LSB=0.1degC **[]Power Mode (cmd_code 0x5): 3-bits, LSB aligned: [list] []**0x0 - Maximum Efficiency **[]0x3 - Maximum Power []**0x4-0x7 - Manufacturer-specific [/list] **[]AVSBus Status (cmd_code 0xe): 16-bits, encoded per AVS Spec []**AVSBus Version (cmd_code 0xf): 4 bits encoded per AVS Spec, LSB-aligned [*]If master issued a write (rw!=0x3) in previous master subframe, this field is 0xffff [/list] |
| 7:3 | Reserved | - | - | Reserved |
| 2:0 | CRC | R RMOD | 0x0 | The CRC-3 code received from slave |

### AVS_DEBUG_READBACK

*Table 2200. avsbus_controller.AVS_DEBUG_READBACK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | AVS_SLAVE_SUBFRAME | R | 0xFFFFFFFF | The current top-of-fifo AVS slave response sub-frame in the readback fifo |

### AVS_LATEST_SLAVE_SUBFRAME

*Table 2201. avsbus_controller.AVS_LATEST_SLAVE_SUBFRAME fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | AVS_SLAVE_SUBFRAME | R | 0xFFFF | The most recently received AVS slave response sub-frame |

### AVS_NORMAL_STATUS

*Table 2202. avsbus_controller.AVS_NORMAL_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:23 | Reserved | - | - | Reserved |
| 22 | AVS_SLAVE_IS_IN_RESYNC | R | 0x0 | Indicates that the AVS master is issuing a slave resync operation (35 AVS clock cycles of holding mdata high) |
| 21 | AVS_BUS_IS_IDLE | R | 0x0 | Indicates that neither AVS master nor slave is currently driving a transaction on the AVS bus |
| 20 | READBACK_HAS_DATA | R | 0x0 | Indicates AVS_READBACK register has data ready to read on APB |
| 19 | READBACK_FIFO_FULL | R | 0x0 | Indicates read-back fifo is full - no new AVS commands will be launched until space is freed by an AVS_READBACK access |
| 18 | CMD_FIFO_FULL | R | 0x0 | Indicates AVS_CMD fifo is full - any additional APB writes to AVS_CMD will be dropped until the AVS bus can accept the next command, thus freeing space in the cmd fifo |
| 17 | CMD_FIFO_EMPTY | R | 0x0 | Indicates AVS_CMD fifo is empty |
| 16 | AVS_MASTER_IS_RETRYING | R | 0x0 | Master is currently in the middle of retrying a failed command |
| 15:0 | TOTAL_RETRIES | R | 0x0 | Total number of retries attempted by the AVS Master |

### AVS_SLAVE_STATUS

*Table 2203. avsbus_controller.AVS_SLAVE_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:18 | Reserved | - | - | Reserved |
| 17:16 | AVS_SLAVE_ACK | R | 0x0 | The SlaveAck bits of the most recent slave subframe received: [list] **[]00: Action Performed []**01:Good CRC received by resource unavailable. No action performed (triggers retry) **[]10: Bad CRC received (triggers retry) []**11: Good CRC received bit bad data, data type, or selector. No action performed [/list] |
| 15:5 | Reserved | - | - | Reserved |
| 4:0 | AVS_SLAVE_STATUS_RESPONSE | R | 0x0 | The 5-bit Slave StatusResponse of the most recent slave subframe received, indicating the general condition of the slave, per AVS spec StatusReponse field: [list] **[]bit 4: VDone - AND of all VDone bits of active rails []**bit 3: StatusAlert - Indicates potential status issue, requiring AVSBus Status read cmd for details **[]bit 2: AVS_Control - 1 if ABSBus is controlling at least 1 device output, 0 otherwise. []**bit 1: MfrSpcfc_Stts1 - Manufacturer-specific [*]bit 0: MfrSpcfc_Stts2 - Manufacturer-specific [/list] |

### AVS_FIFOS_STATUS

*Table 2204. avsbus_controller.AVS_FIFOS_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:24 | `READBACK_FIFO_VACANT_SLOTS` | R | 0x8 | Number of vacant slots in the readback FIFO |
| 23:20 | `Reserved` | - | - | Reserved |
| 19:16 | `READBACK_FIFO_OCCUPIED_SLOTS` | R | 0x0 | Number of occupied slots in the readback FIFO |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:8 | `CMD_FIFO_VACANT_SLOTS` | R | 0x8 | Number of vacant slots in the command FIFO |
| 7:4 | `Reserved` | - | - | Reserved |
| 3:0 | `CMD_FIFO_OCCUPIED_SLOTS` | R | 0x0 | Number of occupied slots in the command FIFO |

### AVS_INTERRUPT

*Table 2205. avsbus_controller.AVS_INTERRUPT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `READBACK_OVERFLOW_INT` | R | 0x0 | Indicates that the response from AVS slave could not be written to readback FIFO due to it being full ⇒ response was dropped |
| 7 | `READBACK_UNDERFLOW_INT` | R | 0x0 | Indicates that an APB read on the AVS_READBACK reg occurred when the readback fifo was empty (AVS_NORMAL_STATUS:readback_has_data=0) |
| 6 | `CMD_FIFO_OVERFLOW_INT` | R | 0x0 | Indicates that an APB write to AVS_CMD was attempted when the the command fifo was already full - the command is dropped |
| 5 | `MAX_RETRIES_ATTEMPTED_INT` | R | 0x0 | Indicates that the AVS Master failed to receive a non-failure acknowledgement from the Slave after a command was retried the maximum allowable number of times (as determined by AVS_CFG_0:max_retries) |
| 4 | `SLAVE_UNRESPONSIVE_INT` | R | 0x0 | Indicates the AVS Slave device is not responding to command subframes from the Master |
| 3 | `READBACK_HAS_DATA_INT` | R | 0x0 | Indicates AVS_READBACK register has data ready to read on APB |
| 2 | `READBACK_FIFO_FULL_INT` | R | 0x0 | Indicates read-back fifo is full - no new AVS commands will be launched until space is freed by an AVS_READBACK access |
| 1 | `CMD_FIFO_FULL_INT` | R | 0x0 | Indicates AVS_CMD fifo is full - any additional APB writes to AVS_CMD will be dropped until the AVS bus can accept the next command, thus freeing space in the cmd fifo |
| 0 | `AVS_SLAVE_ISSUED_INTERRUPT` | R | 0x0 | Indicates the AVS slave has signaled an interrupt |

### AVS_INTERRUPT_MASK

*Table 2206. avsbus_controller.AVS_INTERRUPT_MASK fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `DISABLE_READBACK_OVERFLOW_INT` | RW | 0x1 | Disable readback_overflow interrupt |
| 7 | `DISABLE_READBACK_UNDERFLOW_INT` | RW | 0x1 | Disable readback_underflow interrupt |
| 6 | `DISABLE_CMD_FIFO_OVERFLOW_INT` | RW | 0x1 | Disable cmd_fifo_overflow interrupt |
| 5 | `DISABLE_MAX_RETRIES_ATTEMPTED_INT` | RW | 0x1 | Disable max_retries_attempted interrupt |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 4 | DISABLE_SLAVE_UNRESPONSIVE_INT | RW | 0x1 | Disable slave_unresponsive interrupt |
| 3 | DISABLE_READBACK_HAS_DATA_INT | RW | 0x1 | Disable readback_has_data interrupt |
| 2 | DISABLE_READBACK_FIFO_FULL_INT | RW | 0x1 | Disable readback_fifo_full interrupt |
| 1 | DISABLE_CMD_FIFO_FULL_INT | RW | 0x1 | Disable cmd_fifo_full interrupt |
| 0 | DISABLE_AVS_SLAVE_ISSUED_INTERRUPT | RW | 0x1 | Disable notification of AVS slave interrupt |

### AVS_INTERRUPT_CLEAR

*Table 2207. avsbus_controller.AVS_INTERRUPT_CLEAR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | Reserved | - | - | Reserved |
| 8 | CLEAR_READBACK_OVERFLOW_INT | W 1P | 0x0 | Clear readback_overflow interrupt |
| 7 | CLEAR_READBACK_UNDERFLOW_INT | W 1P | 0x0 | Clear readback_underflow interrupt |
| 6 | CLEAR_CMD_FIFO_OVERFLOW_INT | W 1P | 0x0 | Clear cmd_fifo_overflow interrupt |
| 5 | CLEAR_MAX_RETRIES_ATTEMPTED_INT | W 1P | 0x0 | Clear max_retries_attempted interrupt |
| 4 | CLEAR_SLAVE_UNRESPONSIVE_INT | W 1P | 0x0 | Clear slave_unresponsive interrupt |
| 3 | CLEAR_READBACK_HAS_DATA_INT | W 1P | 0x0 | Clear readback_has_data interrupt |
| 2 | CLEAR_READBACK_FIFO_FULL_INT | W 1P | 0x0 | Clear readback_fifo_full interrupt |
| 1 | CLEAR_CMD_FIFO_FULL_INT | W 1P | 0x0 | Clear cmd_fifo_full interrupt |
| 0 | CLEAR_AVS_SLAVE_ISSUED_INTERRUPT | W 1P | 0x0 | Clear avs_slave_issued_interrupt |

### AVS_CFG_0

*Table 2208. avsbus_controller.AVS_CFG_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |
| 23:16 | MAX_RETRIES | RW | 0x5 | Maximum number of retries, after the original attempt, if an AVS command returns a fail status (either due to targeted resource being busy or CRC failure). A value of 0 disables retries and MAX_RETRIES_ATTEMPTED_INT. |
| 15:0 | RESYNC_INTERVAL | RW | 0x1000 | Interval between periodic slave resync operations on the AVS bus, in APB clock cycles counted from the end of the previous resync. The resulting request then crosses a 3-stage AVS clock synchronizer, and the resync starts once any frame in flight has completed. |

### AVS_CFG_1

*Table 2209. avsbus_controller.AVS_CFG_1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | CLK_DIVIDER_DUTY_CYCLE_NUMERATOR | RW | 0x0 | Determines the desired duty cycle of the divided clock (when avs_clock_select is set to select the divided APB clock or refclk clock). This value will be divided by 256 to determine the high-pulse percentage of the divided clock period. Depending on the clk_divider_value, it may not be possible to precisely achieve the desired duty cycle - in that case, the closest possible fit is used. If this is set to 0, a value of 1 will be assumed. This is set to 0 by default to trick the clock divider into maintaining the HW-default settings (numerator=0x80) by matching the FF default in the resynced version of the signal. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 23:16 | `CLK_DIVIDER_VALUE` | RW | 0x0 | Determines the divisor value used to generate the AVS clock (when avs_clock_select is set to select the divided APB clock or divided refclk clock). The duty cycle of the divided clock depends both on the clk_divider_value together with the clk_divider_duty_cycle_numerator value. A 50% duty cycle is only possible on even clk_divider_values with clk_divider_duty_cycle_numerator set to 128. This field requires a minimum value of 2 - for lower values, 2 will be assumed. This is set to 0 by default to trick the clock divider into maintaining the HW-default settings (divider=0x4) by matching the FF default in the resynced version of the signal. |
| 15:11 | `Reserved` | - | - | Reserved |
| 10 | `TURN_OFF_ALL_PREMUX_CLOCKS` | RW | 0x0 | Gate off all clocks entering the AVS clock mux - do this before changing the mux selection or changing the clock divider settings, then turn back on afterwards |
| 9 | `FORCE_SLAVE_RESYNC_OPERATION` | RW 1P | 0x0 | Force master to issue a slave resync (35 AVS clock cycles of holding the mdata signal high) at the next possible opportunity. The request crosses a 3-stage AVS clock synchronizer before the master sees it. |
| 8 | `STOP_AVS_CLOCK_ON_IDLE` | RW | 0x0 | When the AVS bus is idle, gate the avs_clock from running. Restart it when there are new commands to be run. The master always issues a slave resync first when the clock restarts. |
| 7:2 | `Reserved` | - | - | Reserved |
| 1:0 | `AVS_CLOCK_SELECT` | RW | 0x3 | Selects which clock to use for AVS Clock: [list] **[] 00: Use the APB clock directly as the AVS clock []** 01: Use a divided version of APB clock as the AVS Clock. The clock divider settings are determined by AVS_CFG_1:clk_divider_value and AVS_CFG_1:clk_divider_duty_cycle_numerator **[] 10: Use the refclk clock directly as the AVS clock []** 11: Use a divided version of refclk clock as the AVS Clock. The clock divider settings are determined by AVS_CFG_1:clk_divider_value and AVS_CFG_1:clk_divider_duty_cycle_numerator [/list] |

#### AVS_CONFIG

*Table 2210. avsbus_controller.AVS_CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `AVS_GPIO_ENABLE` | RW | 0x1 | Enable AVSBus connection to GPIOs. The effective sets OE of the GPIOs dedicated to avs_clock and avs_cdata to 1, sets IE of GPIO dedicated to avs_tdata to 1. By default, this is enabled as the AVSBus state machine immediately starts after reset deassertion. |

### GPIO Peripheral

#### Overview

GPIO connects software and low-speed I/O (LSIO) peripherals to individual pads. Software can drive or sample a pad, select an interrupt trigger, and read its boot strap. Hardware LSIO requests take priority over software control unless software explicitly disables LSIO.

Each pad uses two blocks: `gpio` selects data and direction; `gpio_shim` selects electrical settings and captures the strap. Each has its own 32-bit AXI4-Lite register interface. The surrounding padring instantiates these blocks for its pads and supplies the address mapping and LSIO wiring.

The sections below explain ownership and reset behavior, list the interfaces and give the register maps. The GPIO

section of the Programmer's Guide gives the programming sequences.

### Operation

Figure 75 shows the two blocks that serve one pad: the `gpio` controller in the SMC pad ring, and the `gpio_shim` that the adopter's pad integration places in front of the technology pad macro. Each block receives its own AXI-Lite register slot. In `gpio`, the access filter takes its enables and requirements from the `ACCESS_FILTER` fields of `u_gpio_intf_reg` (`write_filter_enable`, `read_filter_enable`, `awprot_requirement` and `arprot_requirement`), the LSIO mux chooses the pad data and enables from the LSIO plane or from `DATA_CTRL`, and the synchronized pad input feeds both the `DATA_CTRL.pad2core` readback and the interrupt detector. In `gpio_shim`, the 2nd-HW override mux can replace the primary plane with the `*_ovrd_i` inputs. While it does, it also returns the pad input on `pad2core_ovrd_o`, which is low otherwise; `pad2core_o` carries the pad input in either case. The electrical mux chooses the drive, pull and glitch-filter settings, and the strap latch captures the pad input at the end of cold reset and presents it both as `CONTROL.strap_value` and on the `captured_strap_o` output. Trapezoids are muxes and demuxes, with their selectable ports on the wide side, so the pad input enters the 2nd-HW override mux on its narrow side; the dashed strap latch is present only when `INPUT_BY_DEFAULT` is set.

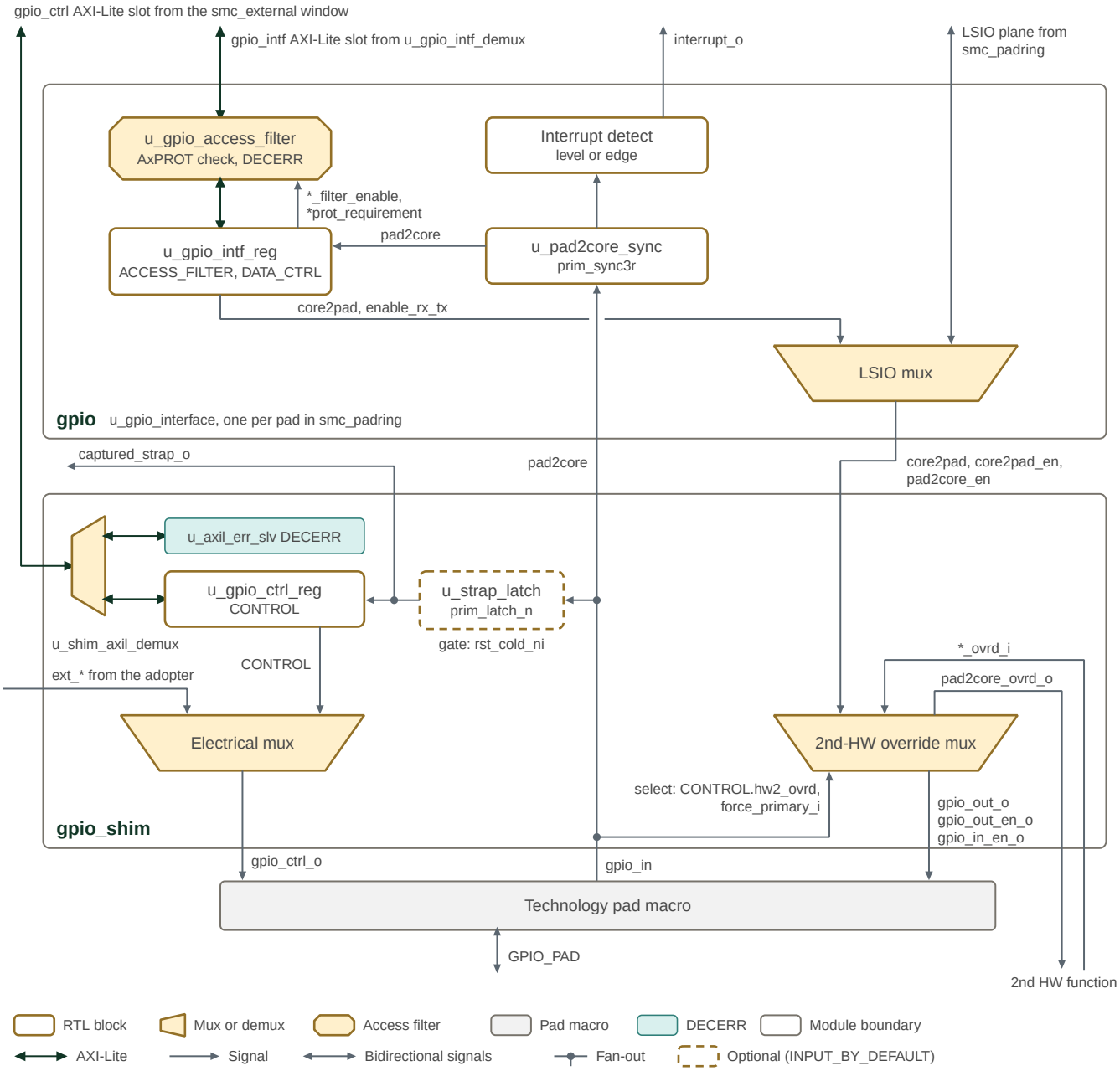


*Figure 75. GPIO controller, interface shim, LSIO mux and strap-capture path*

### Data and Direction Ownership

After cold reset, the controller selects output data, transmit enable and receive enable independently. For each field, the first applicable row wins:

*Table 2211. GPIO data and direction selection*

| Priority | Condition | Source |
|---|---|---|
| 1 | `lsio_interface_select_i = 1` and `DATA_CTRL.lsio_disable = 0` | Hardware LSIO data and direction inputs. |
| 2 | `DATA_CTRL.interface_enable = 1`, or the corresponding `DATA_CTRL_ENABLE.use_reg_*` bit is set | `DATA_CTRL.core2pad`, `enable_rx_tx[0]` (TX), or `enable_rx_tx[1]` (RX). |
| 3 | `DATA_CTRL.lsio_select = 1` and `DATA_CTRL.lsio_disable = 0` | Software-forced LSIO, using the same LSIO inputs. |

| Priority | Condition | Source |
|---|---|---|
| 4 | No preceding condition applies | Output data 0, TX disabled, RX set by `INPUT_BY_DEFAULT`. |

The per-field overrides are `use_reg_core2pad`, `use_reg_tx` and `use_reg_rx`. They allow different sources for data and direction. A hardware LSIO request outranks all three overrides; software-forced LSIO does not. `lsio_disable` gates both forms of LSIO selection.

`DATA_CTRL.lsio_enable` reports the hardware request after the disable gate: `lsio_interface_select_i && !lsio_disable`. It does not report a software-forced selection and is not a general pad-owner indicator.

#### Input Readback

`DATA_CTRL.pad2core` returns the synchronized input when `interface_enable` or `use_reg_rx` selects register readback; otherwise it returns zero. The pad receiver must also be enabled by the selected owner.

The LSIO return path uses the unsynchronized pad input while either LSIO request is active. This return path is independent of the data/direction priority: a software LSIO request can enable readback even when registers own an output field. During cold reset, the LSIO return path also carries the pad input.

#### SMC Pad Ring

The SMC instantiates one `gpio` controller per pad in `u_smc_padring`, 65 in all. Figure 76 shows how the controllers are reached and where the shims sit. `u_smc_padring` sits in `u_smc_peripherals` beside the peripheral crossbar `u_smc_periph_axi_lite_xbar`, which receives the peripheral window from the local crossbar `u_smc_local_xbar` in `u_smc_base`. The peripheral crossbar sends the GPIO interface window to `u_gpio_intf_demux`, which decodes one 16-byte slot per pad and answers any other address with DECERR. The LSIO pad map assigns the SPI, UART, I3C, I2C, AVSBus, system-timer, isolate-request, boot-stall and cool-reset functions to fixed pads, and drives the LSIO plane of each controller. The controllers' pad wires leave the SMC; outside it, the adopter’s pad integration places a `gpio_shim` and a pad macro on each pad, and decodes the `smc_external` window to give each shim its `gpio_ctrl` slot; the rest of that window reaches adopter-specific targets. The figure draws pads 0, 1 and 64; the ellipses stand for the pads between them.

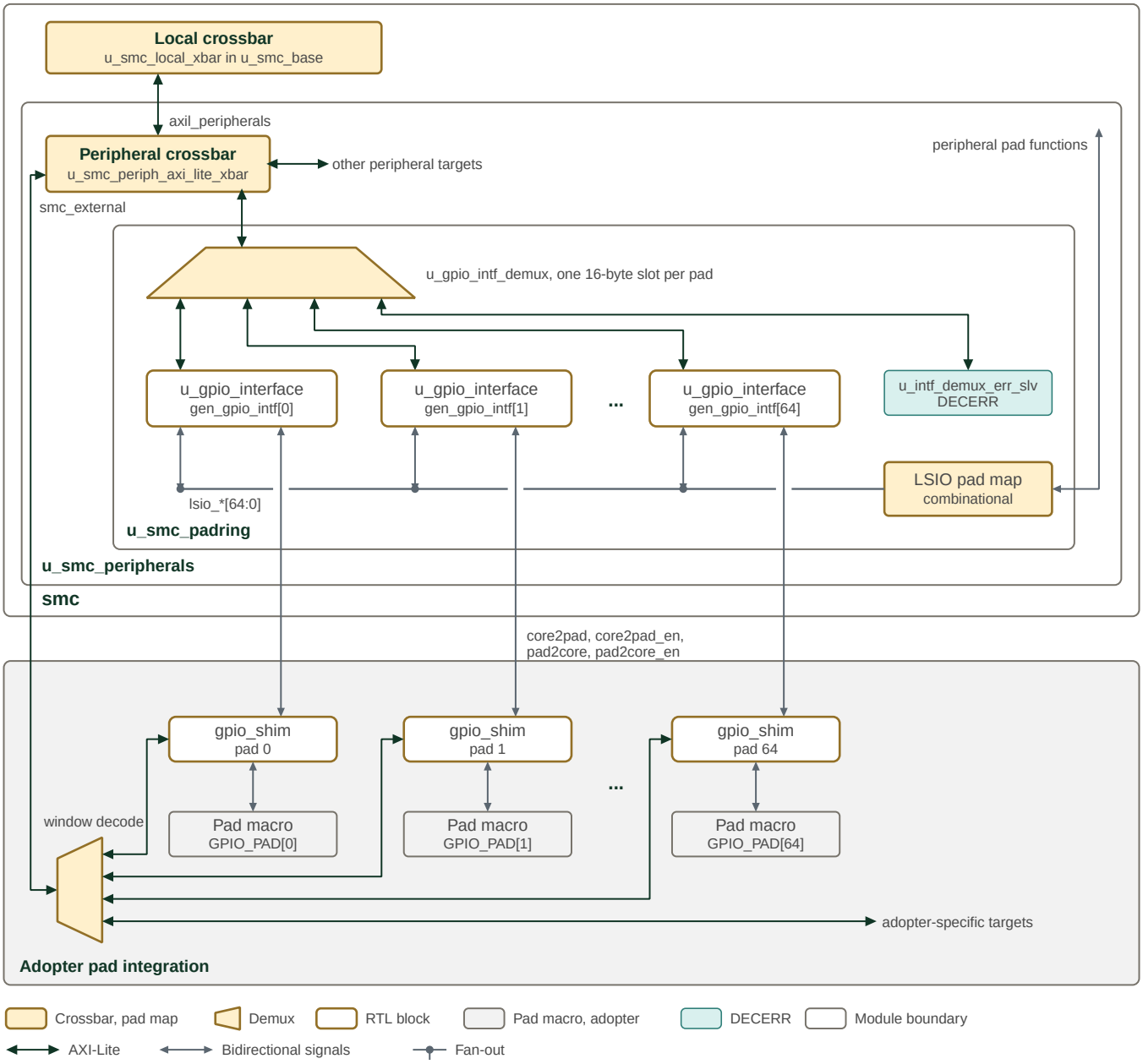


*Figure 76. SMC pad ring GPIO interfaces and adopter pad shims*

The open reference integration, `smc_ip_integration`, stands in for the adopter side with one `prim_pad_shim` per pad at fixed electrical settings, and answers the `gpio_ctrl` range of the `smc_external` window with DECERR.

### Pad Settings and Secondary Function

The shim selects drive strength, pull settings and input conditioning in this order: hardware LSIO request, `CONTROL.config_enable`, software-forced LSIO, then parameter defaults. The LSIO requests use the same disable gate as the controller. In register mode, `CONTROL` supplies `drive_strength`, `pull_enable_n0_scan`, `pull_select` and `schmitt_select`.

A separate mux selects a secondary hardware function when `CONTROL.hw2_ovrd = 1` and `force_primary_i = 0`. It overrides the primary data and direction path. `force_primary_i` restores the primary path regardless of the register setting. The integrator must coordinate switching with both hardware functions.

Electrical effects depend on the connected pad implementation. The supplied `gpio_model` models output drive and

receive enable; it does not model drive strength, pull resistors or input conditioning. Do not interpret the `drive_strength` encoding as a portable current rating.

### Interrupts

The input passes through a three-stage synchronizer before interrupt detection. The selected condition is registered on `clk_i` and gated by `DATA_CTRL.interrupt_enable` at `interrupt_o`.

*Table 2212. GPIO interrupt modes*

| `interrupt_type` | Condition | Output while enabled |
|---|---|---|
| `00` | Input high | Level follows the registered condition. |
| `01` | Input low | Level follows the registered condition. |
| `10` | Rising edge | One clock-cycle pulse per detected edge. |
| `11` | Falling edge | One clock-cycle pulse per detected edge. |

There is no sticky interrupt status or acknowledge register in this block. A level condition must cease, or the interrupt must be disabled, to remove the request. Edge pulses are not stretched; the receiving interrupt controller must capture them. Detection uses the synchronized input, independently of whether register readback is selected.

### Reset and Strap Capture

`rst_primary_ni` asynchronously resets interface/control registers, the input synchronizer and interrupt state. `rst_cold_ni` separately forces controller outputs to data 0, TX disabled and RX=`INPUT_BY_DEFAULT`, and forces the shim's electrical settings to their parameter defaults. The integration supplies reset timing; these modules do not synchronize reset deassertion internally.

With `INPUT_BY_DEFAULT = 1`, the shim's strap latch follows the pad while cold reset is asserted and holds the level when cold reset is released. This is a level-sensitive latch, not a sample taken on a `clk_i` edge. With the parameter clear, the captured value is zero.

`CONTROL.strap_value` and `captured_strap_o` report that captured level. `CONTROL.strap_valid` reflects `INPUT_BY_DEFAULT`; it is not a capture-done flag. Read the strap after the system has released cold reset. Primary reset alone does not recapture it.

### Register Access Filtering

The interface-bank filter compares all three AXI `PROT` bits against the configured read or write requirement. Each direction has its own enable. A mismatch is sent to an error responder and completes with `DECERR`; disabling a filter permits that direction regardless of `PROT`.

Filtering resets disabled. The stored read and write requirements reset to `3'b001` (secure, privileged data), but apply only when enabled. `PROT` expresses transaction attributes, not initiator identity: this filter alone does not establish SEP-only access. It protects the interface bank, not the shim's separate control bank. The GPIO access filtering procedure in the Programmer's Guide gives the setup and readback requirements.

### Interface

The GPIO peripheral provides bus interfaces for register access, LSIO integration for hardware protocols, and direct GPIO pad connections.

Each `gpio` instance drives one pad. The peripheral comprises 65 pads: 61 bonded and 4 unbonded.

The parameters and ports of `gpio` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### LSIO Interface

The LSIO (Low Speed Input/Output) interface enables hardware-controlled GPIO operation for protocol processing.

#### LSIO Protocol Specifications

- **Selection Priority**: an asserted `lsio_interface_select_i` overrides register-driven control, including DATA_CTRL.interface_enable and the DATA_CTRL_ENABLE.use_reg_* per-field overrides. The software force bit DATA_CTRL.lsio_select ranks the other way, below the register interface; DATA_CTRL.lsio_disable gates both
- **Enable Logic**: Active-low enables for both transmit and receive paths
- **Data Latency**: Zero-cycle latency for data path when selected
- **Mode Switching**: Immediate effect, no settling time required
- **Status Reflection**: hardware LSIO request status available in DATA_CTRL.lsio_enable register. The field tracks the hardware request line gated by lsio_disable, so it reads 0 during a software-forced LSIO

#### GPIO Pad Interface

The GPIO pad interface provides direct connection to external pads with bidirectional capabilities.

#### Interrupt Interface

The interrupt interface provides event notification capabilities with configurable trigger modes.

The strap-capture output `captured_strap_o` is a port of `gpio_shim`, not of `gpio`; it holds the pad value captured when the cold reset is released.

#### Interrupt Specifications

- **Polarity**: Active high (asserted when high)
- **Duration**: Level-sensitive, remains asserted based on configuration
- **Trigger Modes**: Level (active-high/low) and edge (rising/falling) detection
- **Clearing**: Automatic clearing based on trigger mode and pad state
- **Latency**: Interrupt assertion within 2-3 clock cycles of triggering event

### Memory Map and Register Reference

#### Memory Map

The controller's interface bank contains `DATA_CTRL`, `DATA_CTRL_ENABLE` and `ACCESS_FILTER`; the shim's control bank contains `CONTROL`. The generated `gpio_intf` and `gpio_ctrl` maps in Detailed Register Map define their register offsets, and `gpio_wrap` defines the bank placement within the composite aperture. The enclosing padring determines their system addresses.

#### Reset Values

`ACCESS_FILTER` resets to `0x00010100`: read and write filters are disabled, with each stored protection requirement set to `3'b001`. `DATA_CTRL` and `DATA_CTRL_ENABLE` software controls reset to zero. Default pad direction and post-reset LSIO

ownership are described in Data and Direction Ownership and Reset and Strap Capture; zeroed software controls do not imply that every pad path is disabled.

`CONTROL` resets its drive-strength field to `3'b010` and its software pull, conditioning, configuration-enable and secondary-override controls to zero. The read-only strap fields reflect `INPUT_BY_DEFAULT` and the captured input, so the complete value is not a single fixed reset word.

### Detailed Register Map

### GPIO Interface Address Map

GPIO interface configuration: pad/core routing, receive and transmit enables, and interrupt control.

*Table 2213. GPIO Interface Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | DATA_CTRL | RW | - |
| 0x4 | DATA_CTRL_ENABLE | RW | - |
| 0x8 | ACCESS_FILTER | RW | GPIO Access Filter Register. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |

### DATA_CTRL

*Table 2214. gpio_intf.DATA_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `pad2core` | R | 0x0 | PAD2SOC Value |
| 30:26 | `Reserved` | - | - | Reserved |
| 25 | `lsio_enable` | R | 0x0 | LSIO hardware request status. Set while lsio_interface_select_i is asserted and lsio_disable is clear. Because the hardware LSIO request outranks interface_enable, a set value also means the LSIO plane is driving this pad's data and direction. This field does not reflect a software-forced selection via lsio_select, which ranks below the register interface - it reads 0 in that case even when LSIO drives the pad. |
| 24:22 | `Reserved` | - | - | Reserved |
| 21:20 | `interrupt_type` | RW | 0x0 | Interrupt type - 0: active-high level, 1: active-low level, 2: rising edge, 3: falling edge |
| 19 | `lsio_disable` | RW | 0x0 | When set, LSIO accesses will be blocked from the GPIO interface |
| 18 | `interrupt_enable` | RW | 0x0 | Interrupt Enable. |
| 17 | `lsio_select` | RW | 0x0 | Force the LSIO interface to be used from software, when no hardware LSIO request is present on this pad. interface_enable has higher priority over this bit. This ranking applies to this software force bit only: the hardware lsio_interface_select_i input outranks interface_enable. Use lsio_disable to gate both. |
| 16 | `interface_enable` | RW | 0x0 | Register Interface Enable. Setting this chooses register values to drive the PAD. This includes core2pad, enable_rx_tx, and pad2core |
| 15:6 | `Reserved` | - | - | Reserved |
| 5:4 | `enable_rx_tx` | RW | 0x0 | Direction control. 2'b00: Neither RX nor TX enabled. 2'b01: TX enabled. 2'b10: RX enabled. 2'b11: Both enabled |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `core2pad` | RW | 0x0 | Register-driven data to send to the pad. |

### DATA_CTRL_ENABLE

*Table 2215. gpio_intf.DATA_CTRL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `use_reg_rx` | RW | 0x0 | Per-field register override. When set, the RX direction (input enable) is taken from enable_rx_tx[1]. OR'd with interface_enable. |
| 4 | `use_reg_tx` | RW | 0x0 | Per-field register override. When set, the TX direction (output enable) is taken from enable_rx_tx[0]. OR'd with interface_enable. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `use_reg_core2pad` | RW | 0x0 | Per-field register override. When set, core2pad is register-driven. OR'd with interface_enable. |

### ACCESS_FILTER

*Table 2216. gpio_intf.ACCESS_FILTER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:19 | `Reserved` | - | - | Reserved |
| 18:16 | `arprot_requirement` | RW | 0x1 | When read_filter_enable is set, only allow read accesses with this prot value WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 15:11 | `Reserved` | - | - | Reserved |
| 10:8 | `awprot_requirement` | RW | 0x1 | When write_filter_enable is set, only allow write accesses with this prot value WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 7:2 | `Reserved` | - | - | Reserved |
| 1 | `read_filter_enable` | RW | 0x0 | Read Filter Enable. When set, reads require an exact match between AXI ARPROT and arprot_requirement. When clear, reads are unfiltered. PROT does not identify the initiator. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 0 | `write_filter_enable` | RW | 0x0 | Write Filter Enable. When set, writes require an exact match between AXI AWPROT and awprot_requirement. When clear, writes are unfiltered. PROT does not identify the initiator. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |

## GPIO Control Address Map

Per-pin GPIO pad control registers: drive strength, pull configuration, and strap capture.

*Table 2217. GPIO Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CONTROL | RW | - |

### CONTROL

*Table 2218. gpio_ctrl.CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | `Reserved` | - | - | Reserved |
| 24 | `hw2_ovrd` | RW | 0x0 | Setting this register will enable the secondary HW function to override the GPIO enable and data lines. |
| 23:22 | `Reserved` | - | - | Reserved |
| 21 | `strap_value` | R | 0x0 | This register holds the captured value that was applied to the pad when reset was de-asserted. |
| 20 | `strap_valid` | R | 0x1 | When set, this bit indicates that the strap value is valid. This field is not set if the GPIO is not an input by default |
| 19:17 | `Reserved` | - | - | Reserved |
| 16 | `config_enable` | RW | 0x0 | Setting this choosing the register values for PAD settings. This includes pull_enable, pull_select, and schmitt_select |
| 15:11 | `Reserved` | - | - | Reserved |
| 10 | `schmitt_select` | RW | 0x0 | Register-driven schmitt select. |
| 9 | `pull_select` | RW | 0x0 | Register-driven pull select - by default we pull down |
| 8 | `pull_enable_n0_scan` | RW | 0x0 | Register-driven pull enable. Default is disabled |
| 7:3 | `Reserved` | - | - | Reserved |
| 2:0 | `drive_strength` | RW | 0x2 | Register-driven drive-strength. |

### GPIO Wrapper Address Map

Top-level GPIO address map composing the interface and control register blocks.

*Table 2219. GPIO Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | DATA_CTRL | RW | - |
| 0x4 | DATA_CTRL_ENABLE | RW | - |
| 0x8 | ACCESS_FILTER | RW | GPIO Access Filter Register. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 0x10 | CONTROL | RW | - |

### DATA_CTRL

*Table 2220. gpio_wrap.DATA_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `pad2core` | R | 0x0 | PAD2SOC Value |
| 30:26 | `Reserved` | - | - | Reserved |
| 25 | `lsio_enable` | R | 0x0 | LSIO hardware request status. Set while lsio_interface_select_i is asserted and lsio_disable is clear. Because the hardware LSIO request outranks interface_enable, a set value also means the LSIO plane is driving this pad's data and direction. This field does not reflect a software-forced selection via lsio_select, which ranks below the register interface - it reads 0 in that case even when LSIO drives the pad. |
| 24:22 | `Reserved` | - | - | Reserved |
| 21:20 | `interrupt_type` | RW | 0x0 | Interrupt type - 0: active-high level, 1: active-low level, 2: rising edge, 3: falling edge |
| 19 | `lsio_disable` | RW | 0x0 | When set, LSIO accesses will be blocked from the GPIO interface |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 18 | `interrupt_enable` | RW | 0x0 | Interrupt Enable. |
| 17 | `lsio_select` | RW | 0x0 | Force the LSIO interface to be used from software, when no hardware LSIO request is present on this pad. interface_enable has higher priority over this bit. This ranking applies to this software force bit only: the hardware lsio_interface_select_i input outranks interface_enable. Use lsio_disable to gate both. |
| 16 | `interface_enable` | RW | 0x0 | Register Interface Enable. Setting this chooses register values to drive the PAD. This includes core2pad, enable_rx_tx, and pad2core |
| 15:6 | `Reserved` | - | - | Reserved |
| 5:4 | `enable_rx_tx` | RW | 0x0 | Direction control. 2'b00: Neither RX nor TX enabled. 2'b01: TX enabled. 2'b10: RX enabled. 2'b11: Both enabled |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `core2pad` | RW | 0x0 | Register-driven data to send to the pad. |

### DATA_CTRL_ENABLE

*Table 2221. gpio_wrap.DATA_CTRL_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `use_reg_rx` | RW | 0x0 | Per-field register override. When set, the RX direction (input enable) is taken from enable_rx_tx[1]. OR'd with interface_enable. |
| 4 | `use_reg_tx` | RW | 0x0 | Per-field register override. When set, the TX direction (output enable) is taken from enable_rx_tx[0]. OR'd with interface_enable. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `use_reg_core2pad` | RW | 0x0 | Per-field register override. When set, core2pad is register-driven. OR'd with interface_enable. |

### ACCESS_FILTER

*Table 2222. gpio_wrap.ACCESS_FILTER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:19 | `Reserved` | - | - | Reserved |
| 18:16 | `arprot_requirement` | RW | 0x1 | When read_filter_enable is set, only allow read accesses with this prot value WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 15:11 | `Reserved` | - | - | Reserved |
| 10:8 | `awprot_requirement` | RW | 0x1 | When write_filter_enable is set, only allow write accesses with this prot value WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |
| 7:2 | `Reserved` | - | - | Reserved |
| 1 | `read_filter_enable` | RW | 0x0 | Read Filter Enable. When set, reads require an exact match between AXI ARPROT and arprot_requirement. When clear, reads are unfiltered. PROT does not identify the initiator. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `write_filter_enable` | RW | 0x0 | Write Filter Enable. When set, writes require an exact match between AXI AWPROT and awprot_requirement. When clear, writes are unfiltered. PROT does not identify the initiator. WARNING: please read back this register to ensure the filter was written correctly. If you do not do so the filter may not update before your next transaction, causing unexpected behavior. |

#### CONTROL

*Table 2223. gpio_wrap.CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | `Reserved` | - | - | Reserved |
| 24 | `hw2_ovrd` | RW | 0x0 | Setting this register will enable the secondary HW function to override the GPIO enable and data lines. |
| 23:22 | `Reserved` | - | - | Reserved |
| 21 | `strap_value` | R | 0x0 | This register holds the captured value that was applied to the pad when reset was de-asserted. |
| 20 | `strap_valid` | R | 0x1 | When set, this bit indicates that the strap value is valid. This field is not set if the GPIO is not an input by default |
| 19:17 | `Reserved` | - | - | Reserved |
| 16 | `config_enable` | RW | 0x0 | Setting this choosing the register values for PAD settings. This includes pull_enable, pull_select, and schmitt_select |
| 15:11 | `Reserved` | - | - | Reserved |
| 10 | `schmitt_select` | RW | 0x0 | Register-driven schmitt select. |
| 9 | `pull_select` | RW | 0x0 | Register-driven pull select - by default we pull down |
| 8 | `pull_enable_n0_scan` | RW | 0x0 | Register-driven pull enable. Default is disabled |
| 7:3 | `Reserved` | - | - | Reserved |
| 2:0 | `drive_strength` | RW | 0x2 | Register-driven drive-strength. |

### Telemetry Receiver

#### Overview

The receiver assembles an 8-bit ATB stream into probe-ID and counter messages. Software reads completed messages through a 32-bit AXI4-Lite register interface and explicitly pops each message. The core defaults to eight queued messages and four counters per message; these are elaboration parameters, not software settings. Threshold and missing-last-packet interrupts report service needs.

Reception does not stop when the message queue fills: a new message replaces the oldest one, including while software is reading it through several registers. The Telemetry Receiver section of the Programmer's Guide gives the buffer, flush and interrupt procedures. The receiver does not use `atid_i` to maintain separate source contexts; producers sharing a receiver must serialize complete messages.

#### Message Format

Eight little-endian ATB beats form a 64-bit packet. One or more packets form a message that ends when `last_packet` is set.

Table 2224 defines the fields carried in each telemetry packet.

*Table 2224. Telemetry packet fields*

| Field | Width | Location |
|---|---|---|
| `last_packet` | 1 bit | Bit 63 of each packet |
| `probe_id` | 5 bits | Bits `[60:56]` of the first packet |
| block | 9 bits | Seven per packet: 1-bit valid + 8-bit data |
| counter | 32 bits | Four consecutive blocks, most-significant byte first |

A counter is valid only when all four of its blocks are valid. The number of counters per message is parameterized (default 4); the field layout is fixed.

#### Architecture

The receiver has two stores: a byte assembly buffer for the incoming message and a queue of decoded messages for software. Neither is a second independently configurable telemetry channel.

#### Assembly and Decode

An accepted beat (`atvalid_i && atready_o`) writes one assembly byte. Eight beats form a packet, least-significant byte first. Bit 63 marks the last packet; the first packet carries the probe ID at bits `[60:56]`. Counter decoding starts at block 1 of the first packet, continues across packet boundaries, and assembles each counter most-significant byte first. All four block-valid bits must be set for a counter to be valid; otherwise its decoded value is zero.

The assembly capacity is derived from `MAX_NUM_COUNTERS_PER_MESSAGE`. Reaching that capacity without a last-packet marker raises `MISSING_LAST` and resets the write pointer. A last-packet marker also resets the pointer and queues the decoded message. The core accepts beats except during a receiver-flush pulse; message-queue fullness does not apply backpressure.

**IMPORTANT**

Every configured counter position is decoded, including positions beyond the last packet received. A short message following a longer one can therefore inherit trailing values and validity bits. Producers must overwrite the complete configured message footprint and mark unused counter blocks invalid. Receiver flush resets the pointer but leaves assembly bytes intact; only reset clears that storage.

#### Message Queue and Readout

The queue exposes its oldest message in `TELEMETRY_PROBE_ID`, `TELEMETRY_COUNTER_VLDS` and `TELEMETRY_COUNTER[0..31]`. Reads do not consume data; `CTRL.BUFFER_POP` advances the queue only when it is nonempty. Empty-queue reads return zero. Counter registers beyond the configured counter count also return zero and have cleared validity bits.

On arrival while full, hardware drops the oldest message and advances the read pointer. There is no snapshot across the separate software reads: overflow can change the visible message mid-read. Arrange sufficient buffering/service rate, or quiesce the producer, when a coherent software snapshot is required. `STATUS` reports empty/full, not a numeric occupancy count or an overflow interrupt.

#### Interrupts

`irq_o` is the OR of the two status sources masked by their corresponding `INTR_ENABLE` bits. Masking an interrupt does not acknowledge its source.

*Table 2225. Telemetry interrupt conditions*

| Source | Condition and clearing |
|---|---|
| `MISSING_LAST` | Sticky status when assembly capacity is reached without the last-packet marker. It latches even while disabled. Write one to the status bit to clear it; `INTR_TEST.MISSING_LAST` supplies a test pulse. |
| `BUFFER_THRESHOLD` | Read-only live status while occupancy is strictly greater than the effective threshold, or `INTR_TEST.BUFFER_THRESHOLD` is set. Drain to at most the threshold, raise the threshold, or clear the test bit as appropriate. Writing the status bit does not clear this source. |

### Reset and Flush

Reset clears assembly bytes, queue pointers and control/interrupt registers. Queue payload storage is not reset, but empty-queue reads return zero. Supply `rst_ni` with asynchronous assertion and deassertion synchronized to `clk_i`.

`CTRL.TELEMETRY_RX_FLUSH` is a pulse that discards queued messages and resets the assembly pointer. It does not erase either payload store or purge the wrapper's upstream ATB crossing FIFO. Quiesce traffic when an empty receiver must remain empty after the pulse.

`CTRL.TELEMETRY_TX_FLUSH` requests a flush from the upstream transmitter through `afvalid_o`. Hardware clears the request after `afvalid_o && afready_i`. Completion depends on the producer's acknowledgement; this is separate from a local receiver flush.

### Interface

The Telemetry Receiver component provides AXI4-Lite register access and Advanced Trace Bus (ATB) telemetry data reception interfaces.

### Configuration Parameters

The parameters and ports of `telemetry_receiver` are listed in the HTML edition of this manual, under RTL Modules Reference.

The core has two elaboration parameters. `BUFFER_DEPTH` (default 8) sets the number of decoded messages the queue holds; it must be a power of two and at least 2. `MAX_NUM_COUNTERS_PER_MESSAGE` (default 4) sets the number of counters decoded from each message.

The ATB beat width (8 bits), packet width (64 bits), counter width (32 bits) and ATB ID width (7 bits) are package constants. The core register port has an 8-bit address and 32-bit data path.

### Signal Interfaces

### Global Interface

The global interface provides fundamental clock and reset signals required for basic operation.

### Clock and Reset

`clk_i` is the core clock and `rst_ni` its asynchronous, active-low reset; `telemetry_receiver_wrap` adds `clk_telemetry_i` / `rst_telemetry_ni` for the ATB side. Clock frequency, jitter, reset duration and physical timing constraints must come from the integration timing specification.

### ATB Telemetry Interface

The ATB (Advanced Trace Bus) interface receives telemetry data using standard ATB protocol.

#### ATB Protocol Specifications

- **Data Width**: 8 bits per beat
- **ID Width**: 7 bits; the receiver ignores `atid_i`
- **Flow Control**: Valid/ready handshaking for reliable data transfer
- **Flush Support**: Bidirectional flush signaling for stream management
- **Back-pressure**: Component can assert `atready_o` low to apply back-pressure
- **Latency**: Zero-cycle latency when ready, immediate acceptance when `atready_o` is high

#### Interrupt Interface

The interrupt interface provides a single interrupt output for event notification.

#### Interrupt Specifications

- **Polarity**: Active high (asserted when high)
- **Duration**: Level-sensitive, remains asserted until cleared
- **Sources**: Buffer threshold and error condition interrupts
- **Clearing**: `MISSING_LAST` is write-one-to-clear. The read-only `BUFFER_THRESHOLD` source remains asserted while occupancy exceeds the configured threshold or its test bit is set; it cannot be cleared by writing status.
- **Latency**: Interrupt assertion within 2 clock cycles of triggering condition

#### Subsystem Wrapper

`telemetry_receiver_wrap` defaults to three receivers. It configures their queue depth with `TELEMETRY_RECEIVER_BUFFER_DEPTH` (default 8) and counter counts with `TELEMETRY_RECEIVER_MAX_NUM_COUNTERS_PER_MESSAGE` (default 4 per receiver). Actual subsystem values and aperture assignments come from the integration.

The wrapper receives ATB traffic on `clk_telemetry_i` / `rst_telemetry_ni` and crosses it into `clk_i` / `rst_ni` through an eight-entry asynchronous FIFO per receiver. Its external ready signal reflects crossing-FIFO capacity. This does not turn the core's decoded-message queue into a lossless backpressured queue. The wrapper also contains flush-handshake synchronizers and returns a decode error for register addresses outside its configured receiver apertures.

### Memory Map and Register Reference

#### Memory Map

Each core occupies 256 bytes with 32-bit registers at four-byte offsets. Control, status, interrupt and message-header registers occupy `0x00`–`0x18`; 32 counter registers occupy `0x80`–`0xFC`. The wrapper map places each receiver in its integration-defined aperture. Reset values and field access types are specified below; reset and flush storage behavior is described in Architecture.

#### Detailed Register Map

#### Telemetry Receiver Address Map

Configuration, status, and data-buffer registers for a single telemetry receiver channel.

| NOTE | **Register Arrays:**<br>• **TELEMETRY_COUNTER[32]**: 32 registers, base 0x80, stride 0x4 |
|---|---|

*Table 2226. Telemetry Receiver Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CTRL | RW | Control Register |
| 0x4 | STATUS | R | Status Register |
| 0x8 | INTR_STATUS | RW | Interrupt Status Register |
| 0xC | INTR_ENABLE | RW | Interrupt Enable Register |
| 0x10 | INTR_TEST | RW | Interrupt Test Register |
| 0x14 | TELEMETRY_PROBE_ID | R | Telemetry Probe ID Register |
| 0x18 | TELEMETRY_COUNTER_VLDS | R | Telemetry Counter Valid Bits Register |
| 0x80 - 0xFC | TELEMETRY_COUNTER[32] | R | Telemetry Counter Register |

### CTRL

*Table 2227. telemetry_receiver.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | `Reserved` | - | - | Reserved |
| 23:12 | `BUFFER_THRESHOLD` | RW | 0x0 | Telemetry Message Buffer Threshold. The Telemetry Buffer Threshold Interrupt is asserted when the number of entries in the buffer is greater than this value. Only the low log2(queue depth)+1 bits are compared, so values alias modulo twice the queue depth, and only values below the queue depth can assert the interrupt. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `TELEMETRY_TX_FLUSH` | RW | 0x0 | Telemetry Transmitter Flush. Writing `1` flushes all FIFOs in the telemetry transmitter. The flushing is done via the ATB AF interface. This bit clears itself after the flush completes. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `TELEMETRY_RX_FLUSH` | W 1P | 0x0 | Telemetry Receiver Flush. Writing `1` discards queued messages and resets the assembly write pointer. Stored payload bytes are not cleared. In particular, a later short message can retain trailing counter values and validity from an earlier message. This operation does not erase storage. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `BUFFER_POP` | W 1P | 0x0 | Telemetry Message Buffer Pop. Effective only if `STATUS.BUFFER_EMPTY = 0`. Writing `1` pops the buffer and thus updates the data in the `TELEMETRY_PROBE_ID`, `TELEMETRY_COUNTER_VLDS`, and `TELEMETRY_COUNTER[32]` registers. |

### STATUS

*Table 2228. telemetry_receiver.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `BUFFER_FULL` | R | 0x0 | Telemetry Message Buffer Full. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `BUFFER_EMPTY` | R | 0x1 | Telemetry Message Buffer Empty. |

### INTR_STATUS

*Table 2229. telemetry_receiver.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `BUFFER_THRESHOLD` | R | 0x0 | Telemetry Message Buffer Threshold Interrupt. Asserted when the number of telemetry messages in the Telemetry Message Buffer is greater than `CTRL.BUFFER_THRESHOLD`. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MISSING_LAST` | RW1C | 0x0 | Last Packet Missing Interrupt. Asserted when the Telemetry Receiver has not received a Last Packet flag when the maximum number of packets a message can consist of has been received. |

### INTR_ENABLE

*Table 2230. telemetry_receiver.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `BUFFER_THRESHOLD` | RW | 0x0 | Telemetry Message Buffer Threshold Interrupt Enable. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MISSING_LAST` | RW | 0x0 | Last Packet Missing Interrupt Enable. |

### INTR_TEST

*Table 2231. telemetry_receiver.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `BUFFER_THRESHOLD` | RW | 0x0 | Telemetry Message Buffer Threshold Interrupt Test. Writing `1` forces the interrupt and writing `0` releases it. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MISSING_LAST` | W 1P | 0x0 | Last Packet Missing Interrupt Test. Writing `1` forces the interrupt. |

### TELEMETRY_PROBE_ID

*Table 2232. telemetry_receiver.TELEMETRY_PROBE_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `PROBE_ID` | R | 0x0 | Probe ID. Contains the Probe ID field of the telemetry message at the bottom of the Telemetry Message Buffer. |

### TELEMETRY_COUNTER_VLDS

*Table 2233. telemetry_receiver.TELEMETRY_COUNTER_VLDS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `COUNTER_VLDS` | R | 0x0 | Counter Valid Bits. Indicates which of the counter values in the telemetry message at the bottom of the Telemetry Message Buffer are valid. Bit `i` is for the `i`-th counter. |

#### TELEMETRY_COUNTER[32]

*Table 2234. telemetry_receiver.TELEMETRY_COUNTER[32] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | COUNTER | R | 0x0 | Counter Value. Contains one of the 32 counter values in the telemetry message at the bottom of the Telemetry Message Buffer. |

### Telemetry Receiver Wrapper Address Map

Wrapper address map instantiating the telemetry receiver channels.

**NOTE**

**Register Arrays:**

- **telemetry_receiver[3].CTRL**: 3 registers, base 0x0, stride 0x100
- **telemetry_receiver[3].STATUS**: 3 registers, base 0x4, stride 0x100
- **telemetry_receiver[3].INTR_STATUS**: 3 registers, base 0x8, stride 0x100
- **telemetry_receiver[3].INTR_ENABLE**: 3 registers, base 0xC, stride 0x100
- **telemetry_receiver[3].INTR_TEST**: 3 registers, base 0x10, stride 0x100
- **telemetry_receiver[3].TELEMETRY_PROBE_ID**: 3 registers, base 0x14, stride 0x100
- **telemetry_receiver[3].TELEMETRY_COUNTER_VLDS**: 3 registers, base 0x18, stride 0x100
- **telemetry_receiver[3].TELEMETRY_COUNTER[32]**: 96 registers, base 0x80 (telemetry_receiver x3 stride 0x100, TELEMETRY_COUNTER x32 stride 0x4)

*Table 2235. Telemetry Receiver Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x200 | telemetry_receiver[3].CTRL | RW | Control Register |
| 0x4 - 0x204 | telemetry_receiver[3].STATUS | R | Status Register |
| 0x8 - 0x208 | telemetry_receiver[3].INTR_STATUS | RW | Interrupt Status Register |
| 0xC - 0x20C | telemetry_receiver[3].INTR_ENABLE | RW | Interrupt Enable Register |
| 0x10 - 0x210 | telemetry_receiver[3].INTR_TEST | RW | Interrupt Test Register |
| 0x14 - 0x214 | telemetry_receiver[3].TELEMETRY_PROBE_ID | R | Telemetry Probe ID Register |
| 0x18 - 0x218 | telemetry_receiver[3].TELEMETRY_COUNTER_VLDS | R | Telemetry Counter Valid Bits Register |
| 0x80 - 0x2FC | telemetry_receiver[3].TELEMETRY_COUNTER[32] | R | Telemetry Counter Register |

#### telemetry_receiver[3].CTRL

*Table 2236. telemetry_receiver_wrap.telemetry_receiver[3].CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 23:12 | BUFFER_THRESHOLD | RW | 0x0 | Telemetry Message Buffer Threshold. The Telemetry Buffer Threshold Interrupt is asserted when the number of entries in the buffer is greater than this value. Only the low log2(queue depth)+1 bits are compared, so values alias modulo twice the queue depth, and only values below the queue depth can assert the interrupt. |
| 11:9 | Reserved | - | - | Reserved |
| 8 | TELEMETRY_TX_FLUSH | RW | 0x0 | Telemetry Transmitter Flush. Writing 1 flushes all FIFOs in the telemetry transmitter. The flushing is done via the ATB AF interface. This bit clears itself after the flush completes. |
| 7:5 | Reserved | - | - | Reserved |
| 4 | TELEMETRY_RX_FLUSH | W 1P | 0x0 | Telemetry Receiver Flush. Writing 1 discards queued messages and resets the assembly write pointer. Stored payload bytes are not cleared. In particular, a later short message can retain trailing counter values and validity from an earlier message. This operation does not erase storage. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | BUFFER_POP | W 1P | 0x0 | Telemetry Message Buffer Pop. Effective only if STATUS.BUFFER_EMPTY = 0. Writing 1 pops the buffer and thus updates the data in the TELEMETRY_PROBE_ID, TELEMETRY_COUNTER_VLDS, and TELEMETRY_COUNTER[32] registers. |

### telemetry_receiver[3].STATUS

*Table 2237. telemetry_receiver_wrap.telemetry_receiver[3].STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | BUFFER_FULL | R | 0x0 | Telemetry Message Buffer Full. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | BUFFER_EMPTY | R | 0x1 | Telemetry Message Buffer Empty. |

### telemetry_receiver[3].INTR_STATUS

*Table 2238. telemetry_receiver_wrap.telemetry_receiver[3].INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | BUFFER_THRESHOLD | R | 0x0 | Telemetry Message Buffer Threshold Interrupt. Asserted when the number of telemetry messages in the Telemetry Message Buffer is greater than CTRL.BUFFER_THRESHOLD. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | MISSING_LAST | RW1C | 0x0 | Last Packet Missing Interrupt. Asserted when the Telemetry Receiver has not received a Last Packet flag when the maximum number of packets a message can consist of has been received. |

### telemetry_receiver[3].INTR_ENABLE

*Table 2239. telemetry_receiver_wrap.telemetry_receiver[3].INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | BUFFER_THRESHOLD | RW | 0x0 | Telemetry Message Buffer Threshold Interrupt Enable. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MISSING_LAST` | RW | 0x0 | Last Packet Missing Interrupt Enable. |

### telemetry_receiver[3].INTR_TEST

*Table 2240. telemetry_receiver_wrap.telemetry_receiver[3].INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `BUFFER_THRESHOLD` | RW | 0x0 | Telemetry Message Buffer Threshold Interrupt Test. Writing `1` forces the interrupt and writing `0` releases it. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `MISSING_LAST` | W 1P | 0x0 | Last Packet Missing Interrupt Test. Writing `1` forces the interrupt. |

### telemetry_receiver[3].TELEMETRY_PROBE_ID

*Table 2241. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_PROBE_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4:0 | `PROBE_ID` | R | 0x0 | Probe ID. Contains the Probe ID field of the telemetry message at the bottom of the Telemetry Message Buffer. |

### telemetry_receiver[3].TELEMETRY_COUNTER_VLDS

*Table 2242. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_COUNTER_VLDS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `COUNTER_VLDS` | R | 0x0 | Counter Valid Bits. Indicates which of the counter values in the telemetry message at the bottom of the Telemetry Message Buffer are valid. Bit `i` is for the `i`-th counter. |

### telemetry_receiver[3].TELEMETRY_COUNTER[32]

*Table 2243. telemetry_receiver_wrap.telemetry_receiver[3].TELEMETRY_COUNTER[32] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `COUNTER` | R | 0x0 | Counter Value. Contains one of the 32 counter values in the telemetry message at the bottom of the Telemetry Message Buffer. |

### UART and Log Engine Wrapper Registers

The wrapper maps include the UART and log-engine control registers and their composed register banks. Offsets are relative to each wrapper's base address.

### UART Wrapper Address Map

Wrapper address map instantiating the per-UART log-engine blocks.

| | |
|---|---|
| **NOTE** | **Register Arrays:** |

- **uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL**: 4 registers, base 0x0, stride 0x400
- **uart_log_engine_wrap[4].uart.RBR**: 4 registers, base 0x100, stride 0x400
- **uart_log_engine_wrap[4].uart.IER**: 4 registers, base 0x104, stride 0x400
- **uart_log_engine_wrap[4].uart.IIR**: 4 registers, base 0x108, stride 0x400
- **uart_log_engine_wrap[4].uart.LCR**: 4 registers, base 0x10C, stride 0x400
- **uart_log_engine_wrap[4].uart.MCR**: 4 registers, base 0x110, stride 0x400
- **uart_log_engine_wrap[4].uart.LSR**: 4 registers, base 0x114, stride 0x400
- **uart_log_engine_wrap[4].uart.MSR**: 4 registers, base 0x118, stride 0x400
- **uart_log_engine_wrap[4].uart.SCR**: 4 registers, base 0x11C, stride 0x400
- **uart_log_engine_wrap[4].uart.ECR**: 4 registers, base 0x120, stride 0x400
- **uart_log_engine_wrap[4].uart.ITR**: 4 registers, base 0x124, stride 0x400
- **uart_log_engine_wrap[4].log_engine.CTRL**: 4 registers, base 0x200, stride 0x400
- **uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE**: 4 registers, base 0x204, stride 0x400
- **uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR**: 4 registers, base 0x208, stride 0x400
- **uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR**: 4 registers, base 0x210, stride 0x400
- **uart_log_engine_wrap[4].log_engine.INTR_STATUS**: 4 registers, base 0x214, stride 0x400
- **uart_log_engine_wrap[4].log_engine.INTR_ENABLE**: 4 registers, base 0x218, stride 0x400
- **uart_log_engine_wrap[4].log_engine.INTR_TEST**: 4 registers, base 0x21C, stride 0x400
- **uart_log_engine_wrap[4].log_engine.LOG_CTRL[16]**: 64 registers, base 0x240 (uart_log_engine_wrap x4 stride 0x400, LOG_CTRL x16 stride 0x4)

*Table 2244. UART Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xC00 | uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL | RW | Control Register |
| 0x100 - 0xD00 | uart_log_engine_wrap[4].uart.RBR | R | Receiver Buffer Register |
| 0x104 - 0xD04 | uart_log_engine_wrap[4].uart.IER | RW | Interrupt Enable Register |
| 0x108 - 0xD08 | uart_log_engine_wrap[4].uart.IIR | R | Interrupt Identification Register |
| 0x10C - 0xD0C | uart_log_engine_wrap[4].uart.LCR | RW | Line Control Register |
| 0x110 - 0xD10 | uart_log_engine_wrap[4].uart.MCR | RW | Modem Control Register |
| 0x114 - 0xD14 | uart_log_engine_wrap[4].uart.LSR | R | Line Status Register |
| 0x118 - 0xD18 | uart_log_engine_wrap[4].uart.MSR | R | Modem Status Register |
| 0x11C - 0xD1C | uart_log_engine_wrap[4].uart.SCR | RW | Scratch Register |
| 0x120 - 0xD20 | uart_log_engine_wrap[4].uart.ECR | RW | Extended Control Register |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x124 - 0xD24 | uart_log_engine_wrap[4].uart.ITR | RW | Interrupt Test Register |
| 0x200 - 0xE00 | uart_log_engine_wrap[4].log_engine.CTRL | RW | Control Register |
| 0x204 - 0xE04 | uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE | RW | Log Region Size Register |
| 0x208 - 0xE08 | uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR | RW | Log Region Address Register |
| 0x210 - 0xE10 | uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR | RW | Log Write Address Register |
| 0x214 - 0xE14 | uart_log_engine_wrap[4].log_engine.INTR_STATUS | RW | Interrupt Status Register |
| 0x218 - 0xE18 | uart_log_engine_wrap[4].log_engine.INTR_ENABLE | RW | Interrupt Enable Register |
| 0x21C - 0xE1C | uart_log_engine_wrap[4].log_engine.INTR_TEST | W | Interrupt Test Register |
| 0x240 - 0xE7C | uart_log_engine_wrap[4].log_engine.LOG_CTRL[16] | RW | Log Control Register |

### uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL

*Table 2245. uart_wrap.uart_log_engine_wrap[4].uart_log_engine_ctrl.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `UART_EN` | RW | 0x0 | UART Enable. When set, the pad-mux downstream will be forced to accept UART traffic. |

### uart_log_engine_wrap[4].uart.RBR

*Table 2246. uart_wrap.uart_log_engine_wrap[4].uart.RBR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R RMOD | 0x0 | Received Data. Contains a received character. If FIFOs are enabled, this field points to the bottom of the RX FIFO. Otherwise, this field points to a single- byte Receiver Buffer Register. |

### uart_log_engine_wrap[4].uart.IER

*Table 2247. uart_wrap.uart_log_engine_wrap[4].uart.IER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `EFEI` | RW | 0x0 | Enable FIFO Error Interrupt. |
| 3 | `EDSSI` | RW | 0x0 | Enable (Delta Status of) Modem Status Interrupt. |
| 2 | `ELSI` | RW | 0x0 | Enable (Receiver) Line Status Interrupt. |
| 1 | `ETBEI` | RW | 0x0 | Enable Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. |
| 0 | `ERBFI` | RW | 0x0 | Enable Receiver Buffer Full (Received Data Ready) Interrupt. |

### uart_log_engine_wrap[4].uart.IIR

*Table 2248. uart_wrap.uart_log_engine_wrap[4].uart.IIR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:6 | `FIFOS_ENABLED` | R | 0x0 | FIFOs Enabled: * `0x0` - FIFOs are disabled * `0x3` - FIFOs are enabled |
| 5:4 | `Reserved` | - | - | Reserved |
| 3:1 | `INTERRUPT_ID` | R | 0x0 | Interrupt ID: * `0x7` - FIFO Error Interrupt (priority 0) * `0x3` - Receiver Line Status Interrupt (priority 1) * `0x6` - Reception Timeout Interrupt (priority 2) * `0x2` - Received Data Ready Interrupt (priority 3) * `0x1` - Transmitter Holding Register Empty Interrupt (priority 4) * `0x0` - Modem Status Interrupt (priority 5) |
| 0 | `INTERRUPT_PENDING` | R | 0x1 | Interrupt Pending. Active-low. |

### uart_log_engine_wrap[4].uart.LCR

*Table 2249. uart_wrap.uart_log_engine_wrap[4].uart.LCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `DLAB` | RW | 0x0 | Divisor Latch Access Bit. If set, allows access to the `DLL` and `DLM` registers when accessing addresses `0x0` and `0x4`, respectively. If unset, allows access to the `THR`, `RBR`, and `IIR` registers. |
| 6 | `SET_BREAK` | RW | 0x0 | Set Break. If set, forces the `tx_o` output to `0` to cause a break condition on the receiving UART. |
| 5 | `STICK_PARITY` | RW | 0x0 | Stick Parity. If set, forces the transmit and received parity bits to be `0` if even parity is selected and `1` if odd parity is selected. |
| 4 | `EPS` | RW | 0x0 | Even Parity Select: * `1` - even parity * `0` - odd parity |
| 3 | `PEN` | RW | 0x0 | Parity Enable. |
| 2 | `STB` | RW | 0x0 | Stop Bits: * `0` - 1 stop bit. * `1` - 2 stop bits (or 1.5 stop bits if word length is set to 5 bits) |
| 1:0 | `WLS` | RW | 0x0 | Word Length Select: * `0x0` - 5 bits per character * `0x1` - 6 bits per character * `0x2` - 7 bits per character * `0x3` - 8 bits per character |

### uart_log_engine_wrap[4].uart.MCR

*Table 2250. uart_wrap.uart_log_engine_wrap[4].uart.MCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |
| 5 | `LINE_LOOPBACK` | RW | 0x0 | Line Loopback. If set, the `rx_i` input is internally connected to the `tx_o` output. |
| 4 | `LOOP` | RW | 0x0 | System Loopback. If set, the transmitter is internally connected to the receiver. The `tx_o` output is set to `1`. |
| 3 | `OUT2` | RW | 0x0 | User Output 2. Writing to this bit drives the `out2_no` in the opposite polarity. |
| 2 | `OUT1` | RW | 0x0 | User Output 1. Writing to this bit drives the `out1_no` in the opposite polarity. |
| 1 | `RTS` | RW | 0x0 | Request to Send. Writing to this bit drives the `rts_no` output in the opposite polarity. |
| 0 | `DTR` | RW | 0x0 | Data Terminal Ready. Writing to this bit drives the `dtr_no` output in the opposite polarity. |

### uart_log_engine_wrap[4].uart.LSR

*Table 2251. uart_wrap.uart_log_engine_wrap[4].uart.LSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `ERROR_IN_RCVR_FIFO` | R | 0x0 | Error in Receiver FIFO. When set, indicates that the receiver FIFO or (RBR in non-FIFO mode) has the parity, framing, or break error. In other words, at least one of the PE, FE, and BI bits is set. |
| 6 | `TEMT` | R | 0x1 | Transmitter Empty. When set, indicates that the transmitter shift register and the transmitter FIFO (or THR in non-FIFO mode) are empty. |
| 5 | `THRE` | R | 0x1 | Transmitter Holding Register Empty. When set, indicates that the TX FIFO (or THR in non-FIFO mode) is empty. Clearing the Transmitter Holding Register Empty Interrupt does NOT clear this bit. |
| 4 | `BI` | R RC | 0x0 | Break Interrupt. When set, indicates that the `rx_i` input is `0` for an entire frame's time (start + data + parity + stop bits). In FIFO mode, this bit set when the character reaches the top of the receiver FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 3 | `FE` | R RC | 0x0 | Framing Error. When set, indicates that the received character is missing a stop bit. In FIFO mode, this bit is set when the character reaches the top of the FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 2 | `PE` | R RC | 0x0 | Parity Error. When set, indicates that the character at the top of the receiver FIFO (or RBR in non-FIFO mode) has a parity error. This bit triggers the Receiver Line Status Interrupt. |
| 1 | `OE` | R RC | 0x0 | Overrun Error. When set, indicates that the RX FIFO (or Receiver Buffer Register in Non-FIFO Mode) is full when new character is received. In FIFO Mode, the new character is dropped. In non-FIFO mode, the existing character in the RBR is overwritten. This bit triggers the Receiver Line Status Interrupt. |
| 0 | `DR` | R | 0x0 | Data Ready. When set, indicates that the RX FIFO (or RBR in non-FIFO mode) contains data. |

### uart_log_engine_wrap[4].uart.MSR

*Table 2252. uart_wrap.uart_log_engine_wrap[4].uart.MSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `DCD` | R | 0x0 | Data Carrier Detect. This bit reflects the `dcd_ni` input in the opposite polarity, so it reads `1` while a carrier is detected. |
| 6 | `RI` | R | 0x0 | Ring Indicator. This bit reflects the `ri_ni` input in the opposite polarity, so it reads `1` while the remote modem signals an incoming call. |
| 5 | `DSR` | R | 0x0 | Data Set Ready. This bit reflects the `dsr_ni` input in the opposite polarity, so it reads `1` while the remote modem is connected and ready. |
| 4 | `CTS` | R | 0x0 | Clear to Send. This bit reflects the `cts_ni` input in the opposite polarity, so it reads `1` while the remote device is ready to receive data. |
| 3 | `DDCD` | R RC | 0x0 | Delta Data Carrier Detect. When set, indicates that the `DCD` bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 2 | `TERI` | R RC | 0x0 | Trailing Edge Ring Indicator. When set, indicates that the `RI` bit has changed from a `1` to a `0`. This bit triggers the Modem Status Interrupt. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | DDSR | R RC | 0x0 | Delta Data Set Ready. When set, indicates that the DSR bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 0 | DCTS | R RC | 0x0 | Delta Clear to Send. When set, indicates that the CTS bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |

### uart_log_engine_wrap[4].uart.SCR

*Table 2253. uart_wrap.uart_log_engine_wrap[4].uart.SCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | SCR | RW | 0x0 | Scratch. Holds user data. |

### uart_log_engine_wrap[4].uart.ECR

*Table 2254. uart_wrap.uart_log_engine_wrap[4].uart.ECR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | Reserved | - | - | Reserved |
| 1:0 | RCVR_TRIGGER_MS2B | RW | 0x0 | Receiver FIFO Trigger Level Most Significant 2 Bits. To configure the least significant 2 bits, use the FCR.RCVR_TRIGGER register field. The configurations for the trigger levels are: * 0x0 - 1 character * 0x1 - 4 characters * 0x2 - 8 characters * 0x3 - 14 characters * 0x4 - 32 characters * 0x5 - 64 characters * 0x6 - 128 characters * 0x7 - 256 characters * 0x8 - 512 characters * 0x9 - 1024 characters * 0xA - 2048 characters * 0xB - 4096 characters |

### uart_log_engine_wrap[4].uart.ITR

*Table 2255. uart_wrap.uart_log_engine_wrap[4].uart.ITR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5 | TRTI | RW | 0x0 | Test Reception Timeout Interrupt. Writing a 1 forces the interrupt and writing 0 releases it. |
| 4 | TFEI | RW | 0x0 | Test FIFO Error Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 3 | TDSSI | RW | 0x0 | Test (Delta Status of) Modem Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 2 | TLSI | RW | 0x0 | Test (Receiver) Line Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 1 | TTBEI | RW | 0x0 | Test Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 0 | TRBFI | RW | 0x0 | Test Receiver Buffer Full (Received Data Ready) Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |

### uart_log_engine_wrap[4].log_engine.CTRL

*Table 2256. uart_wrap.uart_log_engine_wrap[4].log_engine.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | EN | RW | 0x0 | Log Engine Enable. When set, enables the Log Engine. When cleared, the engine stops fetching and writing, discards buffered log data and resets transfer progress. Configuration and pending LOG_LEN values are retained, so a pending entry restarts from the start of its slot when the engine is enabled again. Clearing does not wait for outstanding bus responses. |

### uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE

*Table 2257. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | Reserved | - | - | Reserved |
| 19:0 | LOG_REGION_SIZE | RW | 0x0 | Log region size in bytes. Supported values are at most 512 KiB and aligned to 128 bytes (16 slots times the 8-byte fetch beat). If software programs a larger value, RTL clamps the effective region to 512 KiB. For a value that is not 128-byte aligned, RTL divides the effective region among the 16 slots and rounds each slot's transferable capacity down to complete 8-byte fetch beats. A requested LOG_LEN above that capacity is clamped to the capacity. |

### uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR

*Table 2258. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_REGION_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | RESERVED | R | 0x0 | Reserved. Reads as zero; writes are ignored. |
| 55:32 | LOG_REGION_ADDR_HI | RW | 0x0 | Log Region Start Address High. Contains fabric address bits [55:32]. |
| 31:0 | LOG_REGION_ADDR_LO | RW | 0x0 | Log Region Start Address Low. |

### uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR

*Table 2259. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_WRITE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | LOG_WRITE_ADDR | RW | 0x0 | Log Write Address. This is the address where the log data is written to. |

### uart_log_engine_wrap[4].log_engine.INTR_STATUS

*Table 2260. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | LOG_WRITE_ERR | RW1C | 0x0 | Log Write Error Interrupt. Asserted when the bus for writing log data returns an error. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | LOG_FETCH_ERR | RW1C | 0x0 | Log Fetch Error Interrupt. Asserted when the bus for fetching log data returns an error. |

### uart_log_engine_wrap[4].log_engine.INTR_ENABLE

*Table 2261. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `LOG_WRITE_ERR` | RW | 0x0 | Log Write Error Interrupt Enable. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `LOG_FETCH_ERR` | RW | 0x0 | Log Fetch Error Interrupt Enable. |

### uart_log_engine_wrap[4].log_engine.INTR_TEST

*Table 2262. uart_wrap.uart_log_engine_wrap[4].log_engine.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `LOG_WRITE_ERR` | W 1P | 0x0 | Log Write Error Interrupt Test. Writing `1` forces the interrupt. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `LOG_FETCH_ERR` | W 1P | 0x0 | Log Fetch Error Interrupt Test. Writing `1` forces the interrupt. |

### uart_log_engine_wrap[4].log_engine.LOG_CTRL[16]

*Table 2263. uart_wrap.uart_log_engine_wrap[4].log_engine.LOG_CTRL[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `LOG_LEN` | RW | 0x0 | Log Length. Writing a nonzero value representing the log length in bytes to this field starts the log transfer process. The field holds the requested length until the transfer completes, when hardware clears it to zero; it is not a remaining-byte counter. There are 16 copies of this register, each representing a log entry. The log entries go through round robin arbitration. |

### UART Log Engine Control Address Map

Top-level enable register for the UART log-engine block.

*Table 2264. UART Log Engine Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CTRL | RW | Control Register |

### CTRL

*Table 2265. uart_log_engine_ctrl.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `UART_EN` | RW | 0x0 | UART Enable. When set, the pad-mux downstream will be forced to accept UART traffic. |

### UART Log Engine Wrapper Address Map

Composes the UART 16550 core, log engine, and their control registers for one UART instance.

| NOTE | **Register Arrays:** |
|---|---|

- **LOG_CTRL[16]**: 16 registers, base 0x240, stride 0x4

*Table 2266. UART Log Engine Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | uart_log_engine_ctrl.CTRL | RW | Control Register |
| 0x100 | RBR | R | Receiver Buffer Register |
| 0x104 | IER | RW | Interrupt Enable Register |
| 0x108 | IIR | R | Interrupt Identification Register |
| 0x10C | LCR | RW | Line Control Register |
| 0x110 | MCR | RW | Modem Control Register |
| 0x114 | LSR | R | Line Status Register |
| 0x118 | MSR | R | Modem Status Register |
| 0x11C | SCR | RW | Scratch Register |
| 0x120 | ECR | RW | Extended Control Register |
| 0x124 | ITR | RW | Interrupt Test Register |
| 0x200 | log_engine.CTRL | RW | Control Register |
| 0x204 | LOG_REGION_SIZE | RW | Log Region Size Register |
| 0x208 | LOG_REGION_ADDR | RW | Log Region Address Register |
| 0x210 | LOG_WRITE_ADDR | RW | Log Write Address Register |
| 0x214 | INTR_STATUS | RW | Interrupt Status Register |
| 0x218 | INTR_ENABLE | RW | Interrupt Enable Register |
| 0x21C | INTR_TEST | W | Interrupt Test Register |
| 0x240 - 0x27C | LOG_CTRL[16] | RW | Log Control Register |

### uart_log_engine_ctrl.CTRL

*Table 2267. uart_log_engine_wrap.uart_log_engine_ctrl.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `UART_EN` | RW | 0x0 | UART Enable. When set, the pad-mux downstream will be forced to accept UART traffic. |

### RBR

*Table 2268. uart_log_engine_wrap.RBR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `DATA` | R RMOD | 0x0 | Received Data. Contains a received character. If FIFOs are enabled, this field points to the bottom of the RX FIFO. Otherwise, this field points to a single- byte Receiver Buffer Register. |

### IER

*Table 2269. uart_log_engine_wrap.IER fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |
| 4 | `EFEI` | RW | 0x0 | Enable FIFO Error Interrupt. |
| 3 | `EDSSI` | RW | 0x0 | Enable (Delta Status of) Modem Status Interrupt. |
| 2 | `ELSI` | RW | 0x0 | Enable (Receiver) Line Status Interrupt. |
| 1 | `ETBEI` | RW | 0x0 | Enable Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. |
| 0 | `ERBFI` | RW | 0x0 | Enable Receiver Buffer Full (Received Data Ready) Interrupt. |

### IIR

*Table 2270. uart_log_engine_wrap.IIR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:6 | `FIFOS_ENABLED` | R | 0x0 | FIFOs Enabled: * `0x0` - FIFOs are disabled * `0x3` - FIFOs are enabled |
| 5:4 | `Reserved` | - | - | Reserved |
| 3:1 | `INTERRUPT_ID` | R | 0x0 | Interrupt ID: * `0x7` - FIFO Error Interrupt (priority 0) * `0x3` - Receiver Line Status Interrupt (priority 1) * `0x6` - Reception Timeout Interrupt (priority 2) * `0x2` - Received Data Ready Interrupt (priority 3) * `0x1` - Transmitter Holding Register Empty Interrupt (priority 4) * `0x0` - Modem Status Interrupt (priority 5) |
| 0 | `INTERRUPT_PENDING` | R | 0x1 | Interrupt Pending. Active-low. |

### LCR

*Table 2271. uart_log_engine_wrap.LCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `DLAB` | RW | 0x0 | Divisor Latch Access Bit. If set, allows access to the `DLL` and `DLM` registers when accessing addresses `0x0` and `0x4`, respectively. If unset, allows access to the `THR`, `RBR`, and `IIR` registers. |
| 6 | `SET_BREAK` | RW | 0x0 | Set Break. If set, forces the `tx_o` output to `0` to cause a break condition on the receiving UART. |
| 5 | `STICK_PARITY` | RW | 0x0 | Stick Parity. If set, forces the transmit and received parity bits to be `0` if even parity is selected and `1` if odd parity is selected. |
| 4 | `EPS` | RW | 0x0 | Even Parity Select: * `1` - even parity * `0` - odd parity |
| 3 | `PEN` | RW | 0x0 | Parity Enable. |
| 2 | `STB` | RW | 0x0 | Stop Bits: * `0` - 1 stop bit. * `1` - 2 stop bits (or 1.5 stop bits if word length is set to 5 bits) |
| 1:0 | `WLS` | RW | 0x0 | Word Length Select: * `0x0` - 5 bits per character * `0x1` - 6 bits per character * `0x2` - 7 bits per character * `0x3` - 8 bits per character |

### MCR

*Table 2272. uart_log_engine_wrap.MCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 5 | LINE_LOOPBACK | RW | 0x0 | Line Loopback. If set, the rx_i input is internally connected to the tx_o output. |
| 4 | LOOP | RW | 0x0 | System Loopback. If set, the transmitter is internally connected to the receiver. The tx_o output is set to 1. |
| 3 | OUT2 | RW | 0x0 | User Output 2. Writing to this bit drives the out2_no in the opposite polarity. |
| 2 | OUT1 | RW | 0x0 | User Output 1. Writing to this bit drives the out1_no in the opposite polarity. |
| 1 | RTS | RW | 0x0 | Request to Send. Writing to this bit drives the rts_no output in the opposite polarity. |
| 0 | DTR | RW | 0x0 | Data Terminal Ready. Writing to this bit drives the dtr_no output in the opposite polarity. |

### LSR

*Table 2273. uart_log_engine_wrap.LSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7 | ERROR_IN_RCVR_FIFO | R | 0x0 | Error in Receiver FIFO. When set, indicates that the receiver FIFO or (RBR in non-FIFO mode) has the parity, framing, or break error. In other words, at least one of the PE, FE, and BI bits is set. |
| 6 | TEMT | R | 0x1 | Transmitter Empty. When set, indicates that the transmitter shift register and the transmitter FIFO (or THR in non-FIFO mode) are empty. |
| 5 | THRE | R | 0x1 | Transmitter Holding Register Empty. When set, indicates that the TX FIFO (or THR in non-FIFO mode) is empty. Clearing the Transmitter Holding Register Empty Interrupt does NOT clear this bit. |
| 4 | BI | R RC | 0x0 | Break Interrupt. When set, indicates that the rx_i input is 0 for an entire frame's time (start + data + parity + stop bits). In FIFO mode, this bit set when the character reaches the top of the receiver FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 3 | FE | R RC | 0x0 | Framing Error. When set, indicates that the received character is missing a stop bit. In FIFO mode, this bit is set when the character reaches the top of the FIFO. In non-FIFO mode, this bit is set when the character enters the RBR. This bit triggers the Receiver Line Status Interrupt. |
| 2 | PE | R RC | 0x0 | Parity Error. When set, indicates that the character at the top of the receiver FIFO (or RBR in non-FIFO mode) has a parity error. This bit triggers the Receiver Line Status Interrupt. |
| 1 | OE | R RC | 0x0 | Overrun Error. When set, indicates that the RX FIFO (or Receiver Buffer Register in Non-FIFO Mode) is full when new character is received. In FIFO Mode, the new character is dropped. In non-FIFO mode, the existing character in the RBR is overwritten. This bit triggers the Receiver Line Status Interrupt. |
| 0 | DR | R | 0x0 | Data Ready. When set, indicates that the RX FIFO (or RBR in non-FIFO mode) contains data. |

### MSR

*Table 2274. uart_log_engine_wrap.MSR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7 | DCD | R | 0x0 | Data Carrier Detect. This bit reflects the dcd_ni input in the opposite polarity, so it reads 1 while a carrier is detected. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 6 | RI | R | 0x0 | Ring Indicator. This bit reflects the ri_ni input in the opposite polarity, so it reads 1 while the remote modem signals an incoming call. |
| 5 | DSR | R | 0x0 | Data Set Ready. This bit reflects the dsr_ni input in the opposite polarity, so it reads 1 while the remote modem is connected and ready. |
| 4 | CTS | R | 0x0 | Clear to Send. This bit reflects the cts_ni input in the opposite polarity, so it reads 1 while the remote device is ready to receive data. |
| 3 | DDCD | R RC | 0x0 | Delta Data Carrier Detect. When set, indicates that the DCD bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 2 | TERI | R RC | 0x0 | Trailing Edge Ring Indicator. When set, indicates that the RI bit has changed from a 1 to a 0. This bit triggers the Modem Status Interrupt. |
| 1 | DDSR | R RC | 0x0 | Delta Data Set Ready. When set, indicates that the DSR bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |
| 0 | DCTS | R RC | 0x0 | Delta Clear to Send. When set, indicates that the CTS bit has changed since the last time this register was read. This bit triggers the Modem Status Interrupt. |

### SCR

*Table 2275. uart_log_engine_wrap.SCR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | Reserved | - | - | Reserved |
| 7:0 | SCR | RW | 0x0 | Scratch. Holds user data. |

### ECR

*Table 2276. uart_log_engine_wrap.ECR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | Reserved | - | - | Reserved |
| 1:0 | RCVR_TRIGGER_MS2B | RW | 0x0 | Receiver FIFO Trigger Level Most Significant 2 Bits. To configure the least significant 2 bits, use the FCR.RCVR_TRIGGER register field. The configurations for the trigger levels are: * 0x0 - 1 character * 0x1 - 4 characters * 0x2 - 8 characters * 0x3 - 14 characters * 0x4 - 32 characters * 0x5 - 64 characters * 0x6 - 128 characters * 0x7 - 256 characters * 0x8 - 512 characters * 0x9 - 1024 characters * 0xA - 2048 characters * 0xB - 4096 characters |

### ITR

*Table 2277. uart_log_engine_wrap.ITR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:6 | Reserved | - | - | Reserved |
| 5 | TRTI | RW | 0x0 | Test Reception Timeout Interrupt. Writing a 1 forces the interrupt and writing 0 releases it. |
| 4 | TFEI | RW | 0x0 | Test FIFO Error Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 3 | TDSSI | RW | 0x0 | Test (Delta Status of) Modem Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |
| 2 | TLSI | RW | 0x0 | Test (Receiver) Line Status Interrupt. Writing 1 forces the interrupt and writing 0 releases it. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `TTBEI` | RW | 0x0 | Test Transmitter Buffer Empty (Transmitter Holding Register Empty) Interrupt. Writing `1` forces the interrupt and writing `0` releases it. |
| 0 | `TRBFI` | RW | 0x0 | Test Receiver Buffer Full (Received Data Ready) Interrupt. Writing `1` forces the interrupt and writing `0` releases it. |

### log_engine.CTRL

*Table 2278. uart_log_engine_wrap.log_engine.CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:1 | `Reserved` | - | - | Reserved |
| 0 | `EN` | RW | 0x0 | Log Engine Enable. When set, enables the Log Engine. When cleared, the engine stops fetching and writing, discards buffered log data and resets transfer progress. Configuration and pending LOG_LEN values are retained, so a pending entry restarts from the start of its slot when the engine is enabled again. Clearing does not wait for outstanding bus responses. |

### LOG_REGION_SIZE

*Table 2279. uart_log_engine_wrap.LOG_REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `LOG_REGION_SIZE` | RW | 0x0 | Log region size in bytes. Supported values are at most 512 KiB and aligned to 128 bytes (16 slots times the 8-byte fetch beat). If software programs a larger value, RTL clamps the effective region to 512 KiB. For a value that is not 128-byte aligned, RTL divides the effective region among the 16 slots and rounds each slot's transferable capacity down to complete 8-byte fetch beats. A requested LOG_LEN above that capacity is clamped to the capacity. |

### LOG_REGION_ADDR

*Table 2280. uart_log_engine_wrap.LOG_REGION_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `RESERVED` | R | 0x0 | Reserved. Reads as zero; writes are ignored. |
| 55:32 | `LOG_REGION_ADDR_HI` | RW | 0x0 | Log Region Start Address High. Contains fabric address bits [55:32]. |
| 31:0 | `LOG_REGION_ADDR_LO` | RW | 0x0 | Log Region Start Address Low. |

### LOG_WRITE_ADDR

*Table 2281. uart_log_engine_wrap.LOG_WRITE_ADDR fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `LOG_WRITE_ADDR` | RW | 0x0 | Log Write Address. This is the address where the log data is written to. |

### INTR_STATUS

*Table 2282. uart_log_engine_wrap.INTR_STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 4 | LOG_WRITE_ERR | RW1C | 0x0 | Log Write Error Interrupt. Asserted when the bus for writing log data returns an error. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | LOG_FETCH_ERR | RW1C | 0x0 | Log Fetch Error Interrupt. Asserted when the bus for fetching log data returns an error. |

#### INTR_ENABLE

*Table 2283. uart_log_engine_wrap.INTR_ENABLE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | LOG_WRITE_ERR | RW | 0x0 | Log Write Error Interrupt Enable. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | LOG_FETCH_ERR | RW | 0x0 | Log Fetch Error Interrupt Enable. |

#### INTR_TEST

*Table 2284. uart_log_engine_wrap.INTR_TEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:5 | Reserved | - | - | Reserved |
| 4 | LOG_WRITE_ERR | W 1P | 0x0 | Log Write Error Interrupt Test. Writing 1 forces the interrupt. |
| 3:1 | Reserved | - | - | Reserved |
| 0 | LOG_FETCH_ERR | W 1P | 0x0 | Log Fetch Error Interrupt Test. Writing 1 forces the interrupt. |

#### LOG_CTRL[16]

*Table 2285. uart_log_engine_wrap.LOG_CTRL[16] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | Reserved | - | - | Reserved |
| 15:0 | LOG_LEN | RW | 0x0 | Log Length. Writing a nonzero value representing the log length in bytes to this field starts the log transfer process. The field holds the requested length until the transfer completes, when hardware clears it to zero; it is not a remaining-byte counter. There are 16 copies of this register, each representing a log entry. The log entries go through round robin arbitration. |

### Peripheral Parameter Settings

The SMC sets peripheral parameters at instantiation from `smc_config_pkg`, `smc_pkg`, and the padring package; these are the authoritative sources. Most values match the IP defaults, but the following differ from the default documented in the IP's own section.

Table 2286 compares the SMC parameter overrides with each peripheral's default values.

*Table 2286. SMC Peripheral Parameter Overrides*

| Peripheral | Parameter | IP default | SMC value |
|---|---|---|---|
| Mailbox | smc_pkg::NumMailboxes | 2 | 32 |

| Peripheral | Parameter | IP default | SMC value |
|---|---|---|---|
| Mailbox | `smc_pkg::MailboxDepth` | 8 | 2 |
| I2C | `smc_config_pkg::I2cTargetRxFifoDepth` | 268 | 64 |
| I3C | `smc_config_pkg::NumI3c` | 2 | 6 |
| GPIO | `MAX_TRANS` (threaded from the `smc` module parameter) | 32 | 2 |
| GPIO | `INPUT_BY_DEFAULT` (per instance, from `smc_padring_pkg::DefaultDirectionMap`) | input on all | input on 62 of 65 GPIOs, output on the remaining 3 |
| UART 16550 | `smc_config_pkg::UartTxFifoDepth` | 16 | 32 |
| UART 16550 | `smc_config_pkg::UartRxFifoDepth` | 16 | 32 |

The System Timer OCTS takes no parameters. The eFuse overrides only bus widths and interface types. The Log Engine, AVSBus Controller, and Telemetry Receiver are configured explicitly through `smc_config_pkg`, but at values equal to their IP defaults.

### 2.1.6.8. Design-for-Debug (DFD) and Core Logic Analyzer (CLA)

The SMC integrates a Design-for-Debug (DFD) subsystem that observes internal activity, triggers on programmable conditions, and captures trace, all on-chip and without external instrumentation. Its core is a Core Logic Analyzer (CLA) that watches a configurable debug bus, evaluates programmable events, and fires actions — an interrupt, a clock stop, a cross-trigger, or a trace capture — when those events occur. Trace storage uses the connected external sink-memory interface. Several capabilities of the generic DFD block are not connected in SMC, as detailed below.

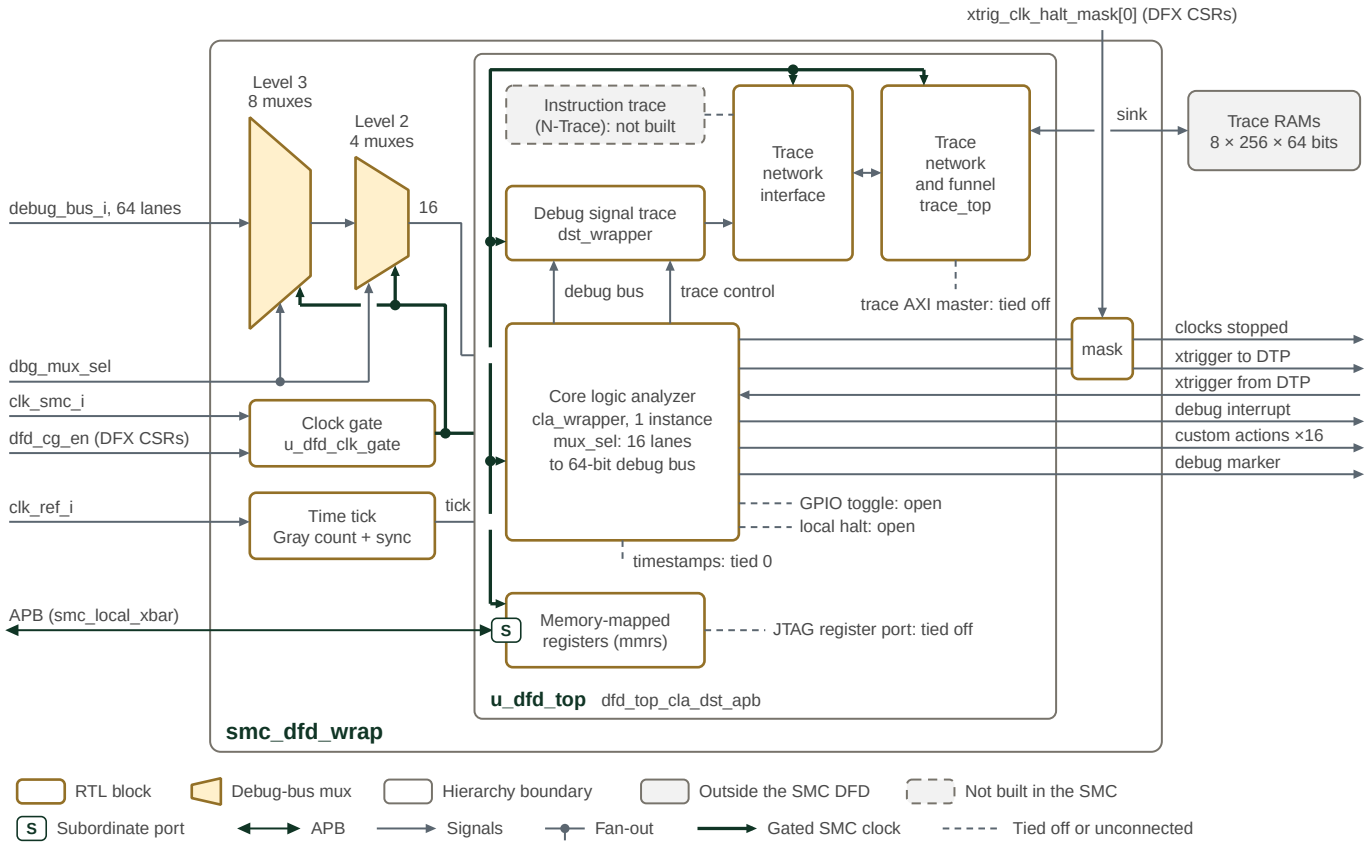


*Figure 77. SMC DFD block diagram*

Figure 77 shows the DFD block as SMC instantiates it. `smc_dfd_wrap` contains the DFD IP `dfd_top_cla_dst_apb` (`u_dfd_top`),

configured with one CLA and one debug signal trace (DST) instance. Parts of the generic block that SMC does not build are drawn as dashed grey boxes. Inputs and outputs that SMC ties off or leaves unconnected are drawn as dashed lines. The instruction trace (N-Trace) is not built. The trace AXI master and the JTAG register port are tied off, and the timestamp inputs are tied to zero. The CLA and DST paths provide the observation functions described here.

The debug bus from `smc_base` has 64 lanes of 16 bits. Eight level-3 and four level-2 debug-bus muxes reduce it to the 16 lanes that enter the CLA, under the select register in the DFX CSRs. The CLA's own mux stage then selects the 64-bit debug bus that the CLA evaluates and passes to the DST. The wrapper gates `clk_smc_i` for the muxes and the IP under `dfd_cg_en`, and it counts `clk_ref_i` edges into the CLA time tick. The trace sink writes eight external trace RAMs of 256 × 64 bits, 16 KiB in total.

Software programs the mux tree one mux at a time through the single `DEBUG_BUS_MUX` register. The level-3 muxes have IDs 7 to 14 and each takes eight consecutive debug-bus lanes; the level-2 muxes have IDs 3 to 6 and each drives four consecutive CLA input lanes. A mux takes `Dbmmode` while `Dbmid` equals its ID, and its lane selects while `Dbmid` equals its ID and its mode is 1. Each lane select is a binary index: 0 passes input lane *n* to output lane *n*, and 1 to 4 select input lanes 4 to 7. The CLA's own stage, programmed through `CDbgMuxSelLo` with `DbmId` 0, decodes its selects the same way, so 1 to 12 select CLA input lanes 4 to 15. The capture set-up sequence is in the Programmer's Guide.

Bit 0 of `xtrig_clk_halt_mask` in the DFX CSRs masks the CLA cross-trigger and halt-clock outputs, with every cross-trigger lane masked together; bits 15:1 have no effect. The clocks-stopped output also requires `tdr_dbg_ctrl_clock_stop_en_i`, and it drives a CPU interrupt and a DTP clock-stop request. The debug interrupt drives a CPU interrupt. The sixteen custom actions leave the SMC, and the SMU wires five of them to the SEP CPU debug controls. The CLA debug marker returns to the SMC debug bus. The `debug_marker`, `chiplet_enable`, `gpio_en`, `dtb_ns_en` and `dtb_ew_en` fields of `DEBUG_CTRL` have no effect in SMC.

Key features include:

- A programmable debug bus that routes internal signals of interest to the analyzer under software control
- An event engine supporting signal-match, edge, transition, ones-count, any-change, and counter events
- Event-action pairing that binds detected events to system actions
- Per-trigger snapshot of the debug bus and status flags identifying which pair fired
- Cross-trigger exchange with the rest of the debug infrastructure
- Trace capture through the external sink-memory interface

#### Debug Bus

The analyzer operates on a debug bus: a fixed-width vector of internal signals sampled every cycle. A selection fabric chooses which internal signals are driven onto the bus, so a single analyzer can be pointed at different parts of the design without RTL changes. Software selects the observed signal set, then programs the event engine against the resulting bus value.

#### Core Logic Analyzer

The CLA is organized as a set of nodes, each holding several event-action pairs (EAPs). An EAP binds one or more events, computed from the debug bus, to one or more actions taken when those events occur. The event engine evaluates several event types in parallel (Table 2287).

*Table 2287. SMC CLA event types*

| Event | Description |
|---|---|
| Signal match | Fires when the masked debug bus equals a programmed value (`debug_bus & mask == match`). Several independent match events are available. |
| Edge | Fires on a rising or falling edge of selected debug-bus signals. |
| Transition | Fires when a masked subset of the bus moves from a programmed value A to a programmed value B. |
| Ones count | Fires when the population count of a masked subset equals a programmed value. |
| Any change | Fires when any signal within a masked subset changes. |
| Time match | Unavailable in this integration. Its timestamp input is tied to zero. The comparator requires a nonzero match value, so the event cannot fire; it does not use the local CLA timestamp register. |
| Counter | Counts cycles following an event match and fires when the count is reached; a small pool of counters is available for measuring intervals or deferring an action. |

When an EAP fires, a status flag records which pair triggered — for example, so an interrupt handler can identify the source — and a snapshot register latches the debug-bus value at the moment of the trigger for post-mortem inspection. EAPs are enabled together once programming is complete, so a partially programmed pairing never triggers.

### Actions and Cross-Triggering

An EAP can drive one or more actions:

- **Debug interrupt** — raise the CLA interrupt, an ordinary maskable PLIC source (see Interrupts below).
- **Clock stop** — halt clocks to freeze the design for inspection.
- **GPIO toggle** — the generic block provides this action, but SMC terminates it on an unused local signal; it does not drive a pad.
- **Cross-trigger out** — signal other debug blocks that a local event occurred.
- **Trace capture** — hand the event to the trace subsystem.
- **Custom actions** — reserved outputs for design-specific use.

Cross-trigger inputs let events elsewhere in the system arm or fire local actions, and cross-trigger outputs propagate local events outward, so debug conditions can be correlated across blocks rather than observed in isolation.

### Trace

The configured trace sink uses external memory request/response ports. The optional trace AXI master has unconnected request outputs and tied-low response inputs, so it cannot stream trace over the SMC fabric.

SMC selects local CLA timestamp scheme 0, advanced by the wrapper's reference clock-derived `time_tick`. This local counter is distinct from the generic `i_timestamp`, reference timestamp, and OCTS timestamp inputs, all tied to zero. The integration does not provide the advertised global timestamp alignment through those inputs. In particular, the time-match event consumes the tied-off timestamp rather than the running CLA timestamp register.

Because of timestamp scheme 0 and the 64-bit CLA debug bus, several CLA registers have no effect in SMC. `TsCapture`, `CDbgTimestampCapture` and `CDbgClaTimestampOffset` serve only the reference-timestamp scheme. The Hi halves of the match, compare, transition, ones-count, change and snapshot registers serve only a 128-bit debug bus. `CDbgMuxSelHi`, and `Muxselseg4` to `Muxselseg7` of `CDbgMuxSelLo`, select lanes beyond the four that the CLA mux drives.

### Interrupts

The SMC interrupt table lists the two CLA-driven sources. See [interrupts].

- **CLA clock stop** — asserted while the CLA has stopped clocks.
- **CLA debug interrupt** — the debug interrupt raised by a CLA action.

### Memory Map and Register Reference

#### Memory Map

The CLA counter values and timestamp fields are software-writable. Software writes take priority over simultaneous hardware updates; later hardware updates can advance or clear them according to their configured controls. The sink's `trDstRamWPHigh` register likewise accepts software writes. These are live read/write state, not read-only status.

The generated `smc_cla` map is the programmer view of this aperture (see Memory Map ).

#### Detailed Register Map

#### SMC DFD MMR Aperture Address Map

Composite debug (DFD) MMR aperture mapping the DST sink, funnel, CLA, and DST blocks that smc_dfd_wrap instantiates.

*Table 2288. SMC DFD MMR Aperture Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | Trdstramcontrol | RW | - |
| 0x4 | Trdstramimpl | RW | - |
| 0x10 | Trdstramstartlow | RW | - |
| 0x14 | Trdstramstarthigh | RW | - |
| 0x18 | Trdstramlimitlow | RW | - |
| 0x1C | Trdstramlimithigh | RW | - |
| 0x20 | Trdstramwplow | RW | - |
| 0x24 | Trdstramwphigh | RW | - |
| 0x28 | Trdstramrplow | RW | - |
| 0x2C | Trdstramrphigh | RW | - |
| 0x40 | Trdstramdata | R | - |
| 0xFF8 | dst_sink.ScratchLo | RW | Scratch register (low) for DV and potential ECO usage |
| 0xFFC | dst_sink.ScratchHi | RW | Scratch register (high) for DV and potential ECO usage |
| 0x1000 | Trfunnelcontrol | RW | - |
| 0x1004 | Trfunnelimpl | R | - |
| 0x1008 | Trfunneldisinput | RW | - |
| 0x1FF8 | funnel.ScratchLo | RW | Scratch register (low) for DV and potential ECO usage |
| 0x1FFC | funnel.ScratchHi | RW | Scratch register (high) for DV and potential ECO usage |
| 0x2100 | CDbgClaCounter0Cfg | RW | Configure CLA counter0. One of the 4 counters used for counting cycles after a event match, and trigger a action on match. |
| 0x2108 | CDbgClaCounter1Cfg | RW | Configure CLA counter1. One of the 4 counters used for counting cycles after a event match, and trigger a action on match. |
| 0x2110 | CDbgClaCounter2Cfg | RW | Configure CLA counter2. One of the 4 counters used for counting cycles after a event match, and trigger a action on match. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2118 | CDbgClaCounter3Cfg | RW | Configure CLA counter3. One of the 4 counters used for counting cycles after a event match, and trigger a action on match. |
| 0x2120 | CDbgNode0Eap0 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2128 | CDbgNode0Eap1 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2130 | CDbgNode1Eap0 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2138 | CDbgNode1Eap1 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2140 | CDbgNode2Eap0 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2148 | CDbgNode2Eap1 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2150 | CDbgNode3Eap0 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2158 | CDbgNode3Eap1 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2160 | CDbgSignalMask0Lo | RW | Used to define debug bus match events. The positive match event (0x2) fires when debug_bus & mask = match; the negative match event (0x3) fires when debug_bus & mask != match. |
| 0x2168 | CDbgSignalMatch0Lo | RW | Used to define debug bus match events. The positive match event (0x2) fires when debug_bus & mask = match; the negative match event (0x3) fires when debug_bus & mask != match. |
| 0x2170 | CDbgSignalMask1Lo | RW | Used to define debug bus match events. The positive match event (0x4) fires when debug_bus & mask = match; the negative match event (0x5) fires when debug_bus & mask != match. |
| 0x2178 | CDbgSignalMatch1Lo | RW | Used to define debug bus match events. The positive match event (0x4) fires when debug_bus & mask = match; the negative match event (0x5) fires when debug_bus & mask != match. |
| 0x2180 | CDbgSignalEdgeDetectCfg | RW | Register to configure debug bus for edge triggers. |
| 0x2188 | CDbgEapStatus | RW | Register to indicate if an EAP was activated. There is one status bit per EAP in bits [15:0] (Node0Eap0..Node3Eap3, 4 EAPs per Node across 4 Nodes) and one write-to-clear bit per EAP in bits [47:32] (Node0Eap0W2C..Node3Eap3W2C); write the matching w2c bit to reset a status bit. The register is used by SW to know if an action was taken (ex: NMI ISR needs to know which of the EAP triggered the NMI) |
| 0x2190 | CDbgClaCtrlStatus | RW | Ctrl/Status register to Enable EAP after EAP programming is complete, and read the current node. |
| 0x2198 | CDbgMuxSelLo | RW | Mode, ID, fine-grain time enable and lane selects for the CLA debug-bus mux, which builds the four 16-bit lanes of the CLA's 64-bit debug bus from 16 input segments. The mux ID is 0: the mode is taken while DbmId is 0, and the lane selects while DbmId is 0 and DbmMode is 1. |
| 0x21A0 | CDbgRsvd1 | RW | Reserved Register that can be changed. |
| 0x21A8 | CDbgRsvd2 | RW | Reserved Register that can be changed. |
| 0x21B0 | CDbgTransitionMaskLo | RW | 3 registers to configure "transition event". This register is to select the signals of interest using a mask. |
| 0x21B8 | CDbgTransitionFromValueLo | RW | 3 registers to configure "transition event". The event is triggered on transition from Value A to Value B. This register specifies Value A. |
| 0x21C0 | CDbgTransitionToValueLo | RW | 3 registers to configure "transition event". The event is triggered on transition from Value A to Value B. This register specifies Value B. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x21C8 | CDbgOnesCountMaskLo | RW | 2 registers to configure "ones count" event. This register is to select the signals of interest using a mask. |
| 0x21D0 | CDbgOnesCountValue | RW | 2 registers to configure "ones count" event. Event triggered when the sum of the signals match the value specified in this register. |
| 0x21D8 | CDbgAnyChangeLo | RW | Event triggered when a subset of debug signals change. The mask is used to select the subset of the debug signals. |
| 0x21E0 | CDbgSignalSnapshotNode0Eap0Lo | R | debug Bus Value when Node0 Eap0 Trigger is met. |
| 0x21E8 | CDbgSignalSnapshotNode0Eap1Lo | R | debug Bus Value when Node0 Eap1 Trigger is met. |
| 0x21F0 | CDbgSignalSnapshotNode1Eap0Lo | R | debug Bus Value when Node1 Eap0 Trigger is met. |
| 0x21F8 | CDbgSignalSnapshotNode1Eap1Lo | R | debug Bus Value when Node1 Eap1 Trigger is met. |
| 0x2200 | CDbgSignalSnapshotNode2Eap0Lo | R | debug Bus Value when Node2 Eap0 Trigger is met. |
| 0x2208 | CDbgSignalSnapshotNode2Eap1Lo | R | debug Bus Value when Node2 Eap1 Trigger is met. |
| 0x2210 | CDbgSignalSnapshotNode3Eap0Lo | R | debug Bus Value when Node3 Eap0 Trigger is met. |
| 0x2218 | CDbgSignalSnapshotNode3Eap1Lo | R | debug Bus Value when Node3 Eap1 Trigger is met. |
| 0x2220 | CDbgClaTimeMatch | RW | This value is compared (greater than or equal to the programmed value) against the internal time register value of the core, and is used to generate a time match event for CLA. Needs to be non zero for the time match event signal to trigger. The event will only trigger an action if the EAP programming is done. Requires writing 0 to this register to stop the match event. |
| 0x2228 | CDbgSignalMask2Lo | RW | Used to define debug bus match events. The positive match event (0x1C) fires when debug_bus & mask = match; the negative match event (0x1D) fires when debug_bus & mask != match. |
| 0x2230 | CDbgSignalMatch2Lo | RW | Used to define debug bus match events. The positive match event (0x1C) fires when debug_bus & mask = match; the negative match event (0x1D) fires when debug_bus & mask != match. |
| 0x2238 | CDbgSignalMask3Lo | RW | Used to define debug bus match events. The positive match event (0x1E) fires when debug_bus & mask = match; the negative match event (0x1F) fires when debug_bus & mask != match. |
| 0x2240 | CDbgSignalMatch3Lo | RW | Used to define debug bus match events. The positive match event (0x1E) fires when debug_bus & mask = match; the negative match event (0x1F) fires when debug_bus & mask != match. |
| 0x2248 | CDbgNode0Eap2 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2250 | CDbgNode0Eap3 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2258 | CDbgNode1Eap2 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2260 | CDbgNode1Eap3 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2268 | CDbgNode2Eap2 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2270 | CDbgNode2Eap3 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2278 | CDbgNode3Eap2 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2280 | CDbgNode3Eap3 | RW | CLA Action(s) are tied to events(s) using "event-action pairing" registers. One of the 16 EAPs (4 EAPs per Node, 4 Nodes). |
| 0x2288 | CDbgSignalSnapshotNode0Eap2Lo | R | debug Bus Value when Node0 Eap2 Trigger is met. |
| 0x2290 | CDbgSignalSnapshotNode0Eap3Lo | R | debug Bus Value when Node0 Eap3 Trigger is met. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2298 | CDbgSignalSnapshotNode1Eap2Lo | R | debug Bus Value when Node1 Eap2 Trigger is met. |
| 0x22A0 | CDbgSignalSnapshotNode1Eap3Lo | R | debug Bus Value when Node1 Eap3 Trigger is met. |
| 0x22A8 | CDbgSignalSnapshotNode2Eap2Lo | R | debug Bus Value when Node2 Eap2 Trigger is met. |
| 0x22B0 | CDbgSignalSnapshotNode2Eap3Lo | R | debug Bus Value when Node2 Eap3 Trigger is met. |
| 0x22B8 | CDbgSignalSnapshotNode3Eap2Lo | R | debug Bus Value when Node3 Eap2 Trigger is met. |
| 0x22C0 | CDbgSignalSnapshotNode3Eap3Lo | R | debug Bus Value when Node3 Eap3 Trigger is met. |
| 0x22C8 | CDbgSignalDelayMuxSel | RW | - |
| 0x22D8 | CDbgClaXtriggerTimestretch | RW | This value determines the time stretched pulse duration for the cross trigger. 8 bits for each of the two triggers. |
| 0x22F0 | CDbgClaTimestamp | RW | 64-bit timestamp counter. Increments on time_tick. Can be written directly or loaded from timestamp_sync on xtrigger when resync is enabled. |
| 0x22F8 | CDbgClaTimestampSync | RW | Timestamp sync value. This value is loaded into the timestamp register on xtrigger posedge when resync bit is enabled. |
| 0x2300 | CDbgClaTimestampConfig | RW | Timestamp configuration register. Controls resync behavior and debug marker. |
| 0x2308 | CDbgClaTimestampOffset | RW | Timestamp offset register. The Offset value is subtracted from the timestamp before comparing with incoming timestamp value. No effect in the SMC, where the CLA timestamp counts the local time tick (timestamp scheme 0). |
| 0x2400 | CDbgSignalMask0Hi | RW | Used to define debug bus match events. The positive match event (0x2) fires when debug_bus & mask = match; the negative match event (0x3) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2408 | CDbgSignalMatch0Hi | RW | Used to define debug bus match events. The positive match event (0x2) fires when debug_bus & mask = match; the negative match event (0x3) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2410 | CDbgSignalMask1Hi | RW | Used to define debug bus match events. The positive match event (0x4) fires when debug_bus & mask = match; the negative match event (0x5) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2418 | CDbgSignalMatch1Hi | RW | Used to define debug bus match events. The positive match event (0x4) fires when debug_bus & mask = match; the negative match event (0x5) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2420 | CDbgSignalMask2Hi | RW | Used to define debug bus match events. The positive match event (0x1C) fires when debug_bus & mask = match; the negative match event (0x1D) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2428 | CDbgSignalMatch2Hi | RW | Used to define debug bus match events. The positive match event (0x1C) fires when debug_bus & mask = match; the negative match event (0x1D) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2430 | CDbgSignalMask3Hi | RW | Used to define debug bus match events. The positive match event (0x1E) fires when debug_bus & mask = match; the negative match event (0x1F) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2438 | CDbgSignalMatch3Hi | RW | Used to define debug bus match events. The positive match event (0x1E) fires when debug_bus & mask = match; the negative match event (0x1F) fires when debug_bus & mask != match. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x2440 | CDbgMuxSelHi | RW | No effect in the SMC. The CLA debug-bus mux has four output lanes and takes its mode, ID, fine-grain time enable and lane selects from CDbgMuxSelLo. |
| 0x2448 | CDbgTransitionMaskHi | RW | 3 registers to configure "transition event". This register is to select the signals of interest using a mask. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2450 | CDbgTransitionFromValueHi | RW | 3 registers to configure "transition event". The event is triggered on transition from Value A to Value B. This register specifies Value A. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2458 | CDbgTransitionToValueHi | RW | 3 registers to configure "transition event". The event is triggered on transition from Value A to Value B. This register specifies Value B. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2460 | CDbgOnesCountMaskHi | RW | 2 registers to configure "ones count" event. This register is to select the signals of interest using a mask. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2468 | CDbgAnyChangeHi | RW | Event triggered when a subset of debug signals change. The mask is used to select the subset of the debug signals. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2470 | CDbgSignalSnapshotNode0Eap0Hi | R | debug Bus Value when Node0 Eap0 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x2478 | CDbgSignalSnapshotNode0Eap1Hi | R | debug Bus Value when Node0 Eap1 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x2480 | CDbgSignalSnapshotNode1Eap0Hi | R | debug Bus Value when Node1 Eap0 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x2488 | CDbgSignalSnapshotNode1Eap1Hi | R | debug Bus Value when Node1 Eap1 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x2490 | CDbgSignalSnapshotNode2Eap0Hi | R | debug Bus Value when Node2 Eap0 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x2498 | CDbgSignalSnapshotNode2Eap1Hi | R | debug Bus Value when Node2 Eap1 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24A0 | CDbgSignalSnapshotNode3Eap0Hi | R | debug Bus Value when Node3 Eap0 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24A8 | CDbgSignalSnapshotNode3Eap1Hi | R | debug Bus Value when Node3 Eap1 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24B0 | CDbgSignalSnapshotNode0Eap2Hi | R | debug Bus Value when Node0 Eap2 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24B8 | CDbgSignalSnapshotNode0Eap3Hi | R | debug Bus Value when Node0 Eap3 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24C0 | CDbgSignalSnapshotNode1Eap2Hi | R | debug Bus Value when Node1 Eap2 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24C8 | CDbgSignalSnapshotNode1Eap3Hi | R | debug Bus Value when Node1 Eap3 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24D0 | CDbgSignalSnapshotNode2Eap2Hi | R | debug Bus Value when Node2 Eap2 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24D8 | CDbgSignalSnapshotNode2Eap3Hi | R | debug Bus Value when Node2 Eap3 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24E0 | CDbgSignalSnapshotNode3Eap2Hi | R | debug Bus Value when Node3 Eap2 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |
| 0x24E8 | CDbgSignalSnapshotNode3Eap3Hi | R | debug Bus Value when Node3 Eap3 Trigger is met. Never captured in the SMC, whose CLA evaluates a 64-bit debug bus. |

| Address | Name | Access | Description |
|---|---|---|---|
| 0x24F0 | CDbgLfsr | RW | LFSR value. |
| 0x24F8 | CDbgLfsrMask | RW | LFSR programming mask. |
| 0x2500 | CDbgTimestampCapture | R | Captured timestamp upper bits. When ts_capture is set in the timestamp config register, a cross trigger 0 event captures the timestamp upper bits into this register. Never captured in the SMC, where the CLA timestamp counts the local time tick (timestamp scheme 0). |
| 0x2508 | CDbgCompare0Lo | RW | Value for arithmetic comparison. |
| 0x2510 | CDbgCompare0MaskLo | RW | Value to mask the arithmetic comparison. |
| 0x2518 | CDbgCompare1Lo | RW | Value for arithmetic comparison. |
| 0x2520 | CDbgCompare1MaskLo | RW | Value to mask the arithmetic comparison. |
| 0x2528 | CDbgCompare2Lo | RW | Value for arithmetic comparison. |
| 0x2530 | CDbgCompare2MaskLo | RW | Value to mask the arithmetic comparison. |
| 0x2538 | CDbgCompare3Lo | RW | Value for arithmetic comparison. |
| 0x2540 | CDbgCompare3MaskLo | RW | Value to mask the arithmetic comparison. |
| 0x2548 | CDbgCompare0Hi | RW | Value for arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2550 | CDbgCompare0MaskHi | RW | Value to mask the arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2558 | CDbgCompare1Hi | RW | Value for arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2560 | CDbgCompare1MaskHi | RW | Value to mask the arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2568 | CDbgCompare2Hi | RW | Value for arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2570 | CDbgCompare2MaskHi | RW | Value to mask the arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2578 | CDbgCompare3Hi | RW | Value for arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2580 | CDbgCompare3MaskHi | RW | Value to mask the arithmetic comparison. No effect in the SMC, whose CLA evaluates a 64-bit debug bus; only the Lo register is used. |
| 0x2FF8 | Scratch | RW | Additional scratch register for DV and potential ECO usage |
| 0x3000 | Trdstcontrol | RW | - |
| 0x3004 | Trdstimpl | RW | - |
| 0x3008 | Trdstinstfeatures | RW | - |
| 0x31A0 | CDbgDebugTraceCfg | RW | - |
| 0x3FF8 | dst.ScratchLo | RW | Scratch register (low) for DV and potential ECO usage |
| 0x3FFC | dst.ScratchHi | RW | Scratch register (high) for DV and potential ECO usage |

### Trdstramcontrol

*Table 2289. smc_cla.Trdstramcontrol fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:9 | `Reserved` | - | - | Reserved |
| 8 | `Trdstramstoponwrap` | RW | 0x0 | - |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `Trdstrammode` | RW | 0x0 | - |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 3 | `Trdstramempty` | R | 0x1 | - |
| 2 | `Reserved` | - | - | Reserved |
| 1 | `Trdstramenable` | RW | 0x0 | - |
| 0 | `Trdstramactive` | RW | 0x0 | - |

### Trdstramimpl

*Table 2290. smc_cla.Trdstramimpl fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Reserved` | - | - | Reserved |
| 27:24 | `Trdstramvendorframelength` | RW | 0x1 | Frame length in 64-byte units. Only 1, 2, 4 and 8 (64, 128, 256 and 512 bytes) are supported, and the value must match the DST's trDstVendorFrameLength. |
| 23:14 | `Reserved` | - | - | Reserved |
| 13 | `Trdstramhassmem` | R | 0x1 | - |
| 12 | `Trdstramhassram` | R | 0x1 | - |
| 11:8 | `Trdstramcomptype` | R | 0x9 | - |
| 7:4 | `Trdstramverminor` | R | 0x0 | - |
| 3:0 | `Trdstramvermajor` | R | 0x1 | - |

### Trdstramstartlow

*Table 2291. smc_cla.Trdstramstartlow fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Trdstramstartlow` | RW | 0x0 | - |
| 1:0 | `Rsvd10` | R | 0x0 | - |

### Trdstramstarthigh

*Table 2292. smc_cla.Trdstramstarthigh fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Trdstramstarthigh` | RW | 0x0 | - |

### Trdstramlimitlow

*Table 2293. smc_cla.Trdstramlimitlow fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Trdstramlimitlow` | RW | 0x0 | - |
| 1:0 | `Rsvd10` | R | 0x0 | - |

### Trdstramlimithigh

*Table 2294. smc_cla.Trdstramlimithigh fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Trdstramlimithigh` | RW | 0x0 | - |

### Trdstramwplow

*Table 2295. smc_cla.Trdstramwplow fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Trdstramwplow` | RW | 0x0 | - |
| 1 | `Reserved` | - | - | Reserved |
| 0 | `Trdstramwrap` | R | 0x0 | - |

### Trdstramwphigh

*Table 2296. smc_cla.Trdstramwphigh fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Trdstramwphigh` | RW | 0x0 | - |

### Trdstramrplow

*Table 2297. smc_cla.Trdstramrplow fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:2 | `Trdstramrplow` | RW | 0x0 | Read pointer, in 32-bit words; bits [31:2] of the byte address of the word the next trDstRamData read returns. Each trDstRamData read that returns unread data advances it by one word, and from trDstRamLimitLow it wraps to trDstRamStartLow. |
| 1:0 | `Rsvd10` | R | 0x0 | - |

### Trdstramrphigh

*Table 2298. smc_cla.Trdstramrphigh fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Trdstramrphigh` | RW | 0x0 | - |

### Trdstramdata

*Table 2299. smc_cla.Trdstramdata fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Trdstramdata` | R | 0x0 | Reading launches a trace-RAM read at trDstRamRPLow. When the RAM holds unread data, the read returns that 32-bit word and advances trDstRamRPLow; otherwise the pointer does not move. |

### dst_sink.ScratchLo

*Table 2300. smc_cla.dst_sink.ScratchLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

### dst_sink.ScratchHi

*Table 2301. smc_cla.dst_sink.ScratchHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

### Trfunnelcontrol

*Table 2302. smc_cla.Trfunnelcontrol fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:4 | `Reserved` | - | - | Reserved |
| 3 | `Trfunnelempty` | R | 0x1 | - |
| 2 | `Reserved` | - | - | Reserved |
| 1 | `Trfunnelenable` | RW | 0x0 | - |
| 0 | `Trfunnelactive` | RW | 0x0 | - |

### Trfunnelimpl

*Table 2303. smc_cla.Trfunnelimpl fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | `Reserved` | - | - | Reserved |
| 11:8 | `Trfunnelcomptype` | R | 0x8 | - |
| 7:4 | `Trfunnelverminor` | R | 0x0 | e |
| 3:0 | `Trfunnelvermajor` | R | 0x1 | - |

### Trfunneldisinput

*Table 2304. smc_cla.Trfunneldisinput fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `Trfunneldisinput` | RW | 0x0 | Bits 7:0 are reserved for N-trace srcs and 15:8 for Dst sources |

### funnel.ScratchLo

*Table 2305. smc_cla.funnel.ScratchLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

### funnel.ScratchHi

*Table 2306. smc_cla.funnel.ScratchHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

### CDbgClaCounter0Cfg

*Table 2307. smc_cla.CDbgClaCounter0Cfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `Rsvd` | RW | 0x0 | - |
| 62:48 | `UpperTarget` | RW | 0x0 | - |
| 47:33 | `UpperCounter` | RW | 0x0 | - |
| 32 | `ResetOnTarget` | RW | 0x0 | When Counter = {upper_target,target} reset to 0. With this bit set, counter will provide a periodic tick w/ frequency of (target +1) if the action is set to Auto Incr. The periodic tick will continue till Stop Auto Incr or Clear Counter action. |
| 31:16 | `Target` | RW | 0x0 | - |
| 15:0 | `Counter` | RW | 0x0 | - |

### CDbgClaCounter1Cfg

*Table 2308. smc_cla.CDbgClaCounter1Cfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `Rsvd` | RW | 0x0 | - |
| 62:48 | `UpperTarget` | RW | 0x0 | - |
| 47:33 | `UpperCounter` | RW | 0x0 | - |
| 32 | `ResetOnTarget` | RW | 0x0 | When Counter = {upper_target,target}, reset to 0. With this bit set, counter will provide a periodic tick w/ frequency of (target +1) if the action is set to Auto Incr. The periodic tick will continue till Stop Auto Incr or Clear Counter action. |
| 31:16 | `Target` | RW | 0x0 | - |
| 15:0 | `Counter` | RW | 0x0 | - |

### CDbgClaCounter2Cfg

*Table 2309. smc_cla.CDbgClaCounter2Cfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `Rsvd` | RW | 0x0 | - |
| 62:48 | `UpperTarget` | RW | 0x0 | - |
| 47:33 | `UpperCounter` | RW | 0x0 | - |
| 32 | `ResetOnTarget` | RW | 0x0 | When Counter = {upper_target,target}, reset to 0. With this bit set, counter will provide a periodic tick w/ frequency of (target +1) if the action is set to Auto Incr. The periodic tick will continue till Stop Auto Incr or Clear Counter action. |
| 31:16 | `Target` | RW | 0x0 | - |
| 15:0 | `Counter` | RW | 0x0 | - |

### CDbgClaCounter3Cfg

*Table 2310. smc_cla.CDbgClaCounter3Cfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `Rsvd` | RW | 0x0 | - |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 62:48 | `UpperTarget` | RW | 0x0 | - |
| 47:33 | `UpperCounter` | RW | 0x0 | - |
| 32 | `ResetOnTarget` | RW | 0x0 | When Counter = {upper_target,target}, reset to 0. With this bit set, counter will provide a periodic tick w/ frequency of (target +1) if the action is set to Auto Incr. The periodic tick will continue till Stop Auto Incr or Clear Counter action. |
| 31:16 | `Target` | RW | 0x0 | - |
| 15:0 | `Counter` | RW | 0x0 | - |

### CDbgNode0Eap0

*Table 2311. smc_cla.CDbgNode0Eap0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode0Eap1

*Table 2312. smc_cla.CDbgNode0Eap1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode1Eap0

*Table 2313. smc_cla.CDbgNode1Eap0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode1Eap1

*Table 2314. smc_cla.CDbgNode1Eap1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode2Eap0

*Table 2315. smc_cla.CDbgNode2Eap0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode2Eap1

*Table 2316. smc_cla.CDbgNode2Eap1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode3Eap0

*Table 2317. smc_cla.CDbgNode3Eap0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode3Eap1

*Table 2318. smc_cla.CDbgNode3Eap1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgSignalMask0Lo

*Table 2319. smc_cla.CDbgSignalMask0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch0Lo

*Table 2320. smc_cla.CDbgSignalMatch0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalMask1Lo

*Table 2321. smc_cla.CDbgSignalMask1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch1Lo

*Table 2322. smc_cla.CDbgSignalMatch1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalEdgeDetectCfg

*Table 2323. smc_cla.CDbgSignalEdgeDetectCfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:16 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15 | PosEdgeSignal1 | RW | 0x0 | Defines which edge to detect for singal from signal1_select. 1: Pos edge, 0: Neg Edge |
| 14:8 | Signal1Select | RW | 0x0 | Define which signal to select for edge detection. |
| 7 | PosEdgeSignal0 | RW | 0x0 | Defines which edge to detect for singal from signal0_select. 1: Pos edge, 0: Neg Edge |
| 6:0 | Signal0Select | RW | 0x0 | Define which signal to select for edge detection. |

### CDbgEapStatus

*Table 2324. smc_cla.CDbgEapStatus fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:48 | Reserved | - | - | Reserved |
| 47 | Node3Eap3W2C | RW | 0x0 | - |
| 46 | Node3Eap2W2C | RW | 0x0 | - |
| 45 | Node3Eap1W2C | RW | 0x0 | - |
| 44 | Node3Eap0W2C | RW | 0x0 | - |
| 43 | Node2Eap3W2C | RW | 0x0 | - |
| 42 | Node2Eap2W2C | RW | 0x0 | - |
| 41 | Node2Eap1W2C | RW | 0x0 | - |
| 40 | Node2Eap0W2C | RW | 0x0 | - |
| 39 | Node1Eap3W2C | RW | 0x0 | - |
| 38 | Node1Eap2W2C | RW | 0x0 | - |
| 37 | Node1Eap1W2C | RW | 0x0 | - |
| 36 | Node1Eap0W2C | RW | 0x0 | - |
| 35 | Node0Eap3W2C | RW | 0x0 | - |
| 34 | Node0Eap2W2C | RW | 0x0 | - |
| 33 | Node0Eap1W2C | RW | 0x0 | - |
| 32 | Node0Eap0W2C | RW | 0x0 | - |
| 31:16 | Rsvd3116 | R | 0x0 | - |
| 15 | Node3Eap3 | R | 0x0 | - |
| 14 | Node3Eap2 | R | 0x0 | - |
| 13 | Node3Eap1 | R | 0x0 | - |
| 12 | Node3Eap0 | R | 0x0 | - |
| 11 | Node2Eap3 | R | 0x0 | - |
| 10 | Node2Eap2 | R | 0x0 | - |
| 9 | Node2Eap1 | R | 0x0 | - |
| 8 | Node2Eap0 | R | 0x0 | - |
| 7 | Node1Eap3 | R | 0x0 | - |
| 6 | Node1Eap2 | R | 0x0 | - |
| 5 | Node1Eap1 | R | 0x0 | - |
| 4 | Node1Eap0 | R | 0x0 | - |
| 3 | Node0Eap3 | R | 0x0 | - |
| 2 | Node0Eap2 | R | 0x0 | - |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 1 | `Node0Eap1` | R | 0x0 | - |
| 0 | `Node0Eap0` | R | 0x0 | - |

### CDbgClaCtrlStatus

*Table 2325. smc_cla.CDbgClaCtrlStatus fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63 | `ClaLock` | RW | 0x0 | If this bit is set, then CLA is locked and the enable_cla is latched forever |
| 62:16 | `Rsvd6216` | RW | 0x0 | reserved fields |
| 15 | `DisableLocalClockHalt` | RW | 0x0 | If this bit is set, clock halt action is not applied to local clocks |
| 14 | `DisableGlobalClockHalt` | RW | 0x0 | If this bit is set, clock halt action is not applied to global clocks |
| 13:7 | `ClaChainLoopDelay` | RW | 0x36 | Self-filter delay, in CLA clock cycles, for the CLA's own cross-triggers. When an EAP action drives cross-trigger output n, the CLA ignores incoming cross-trigger n for one cycle, ClaChainLoopDelay cycles later, so that its own trigger returning through the cross-trigger network does not retrigger it; with 0 it ignores the input in the same cycle. A further own trigger while the delay counts does not restart it. Incoming clock halt is not filtered. |
| 6 | `EnableCla` | RW | 0x0 | Set this bit to 1 to enable the CLA functional clock; clear it to 0 to gate that clock (save pwr). The CLA register block runs on a clock whose functional enable is tied high, so this bit can still be written while the functional clock is gated |
| 5 | `EnableEap` | RW | 0x0 | Set Enable EAP to 1 after EAP programming is complete |
| 4:2 | `Reserved` | - | - | Reserved |
| 1:0 | `CurrentNode` | R | 0x0 | Read the current node. |

### CDbgMuxSelLo

*Table 2326. smc_cla.CDbgMuxSelLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Muxselseg7` | RW | 0x0 | No effect in the SMC: the CLA debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 57:52 | `Muxselseg6` | RW | 0x0 | No effect in the SMC: the CLA debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 51:46 | `Muxselseg5` | RW | 0x0 | No effect in the SMC: the CLA debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 45:40 | `Muxselseg4` | RW | 0x0 | No effect in the SMC: the CLA debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 39:34 | `Muxselseg3` | RW | 0x0 | Binary segment select for lane 3 of the CLA debug bus, bits [63:48]. The input segments are 16 bits wide: Seg0 = input bits [15:0], Seg1 = [31:16], and so on to Seg15. 0 selects Seg3; a value k from 1 to 12 selects Seg(k+3), so 1 selects Seg4 and 4 selects Seg7; 13 to 63 drive zero. When FineGrainTime is set, the fine-grain time counter is inserted below Seg0: 0 selects Seg2, and k from 1 to 13 selects Seg(k+2). |
| 33:28 | `Muxselseg2` | RW | 0x0 | Binary segment select for lane 2 of the CLA debug bus, bits [47:32]. The input segments are 16 bits wide: Seg0 = input bits [15:0], Seg1 = [31:16], and so on to Seg15. 0 selects Seg2; a value k from 1 to 12 selects Seg(k+3), so 1 selects Seg4 and 4 selects Seg7; 13 to 63 drive zero. When FineGrainTime is set, the fine-grain time counter is inserted below Seg0: 0 selects Seg1, and k from 1 to 13 selects Seg(k+2). |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 27:22 | `Muxselseg1` | RW | 0x0 | Binary segment select for lane 1 of the CLA debug bus, bits [31:16]. The input segments are 16 bits wide: Seg0 = input bits [15:0], Seg1 = [31:16], and so on to Seg15. 0 selects Seg1; a value k from 1 to 12 selects Seg(k+3), so 1 selects Seg4 and 4 selects Seg7; 13 to 63 drive zero. When FineGrainTime is set, the fine-grain time counter is inserted below Seg0: 0 selects Seg0, and k from 1 to 13 selects Seg(k+2). |
| 21:16 | `Muxselseg0` | RW | 0x0 | Binary segment select for lane 0 of the CLA debug bus, bits [15:0]. The input segments are 16 bits wide: Seg0 = input bits [15:0], Seg1 = [31:16], and so on to Seg15. 0 selects Seg0; a value k from 1 to 12 selects Seg(k+3), so 1 selects Seg4 and 4 selects Seg7; 13 to 63 drive zero. When FineGrainTime is set, the fine-grain time counter is inserted below Seg0: 0 selects the fine-grain time counter, and k from 1 to 13 selects Seg(k+2). |
| 15:9 | `Rsvd159` | RW | 0x0 | - |
| 8 | `Finegraintime` | RW | 0x0 | Fine Grain Time Enable. If this bit is set, the fine grain time is enabled. |
| 7:2 | `Dbmid` | RW | 0x0 | Unique DBM ID of the DBM instance |
| 1:0 | `Dbmmode` | RW | 0x0 | Mode selection, 0: DBM off, 1: Normal debug mode, 2: DBM ID output mode, 3: Toggle Mode |

### CDbgRsvd1

*Table 2327. smc_cla.CDbgRsvd1 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value of reserved register. |

### CDbgRsvd2

*Table 2328. smc_cla.CDbgRsvd2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value of reserved register. |

### CDbgTransitionMaskLo

*Table 2329. smc_cla.CDbgTransitionMaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgTransitionFromValueLo

*Table 2330. smc_cla.CDbgTransitionFromValueLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgTransitionToValueLo

*Table 2331. smc_cla.CDbgTransitionToValueLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgOnesCountMaskLo

*Table 2332. smc_cla.CDbgOnesCountMaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgOnesCountValue

*Table 2333. smc_cla.CDbgOnesCountValue fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgAnyChangeLo

*Table 2334. smc_cla.CDbgAnyChangeLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Mask` | RW | 0x0 | - |

### CDbgSignalSnapshotNode0Eap0Lo

*Table 2335. smc_cla.CDbgSignalSnapshotNode0Eap0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode0Eap1Lo

*Table 2336. smc_cla.CDbgSignalSnapshotNode0Eap1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap0Lo

*Table 2337. smc_cla.CDbgSignalSnapshotNode1Eap0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap1Lo

*Table 2338. smc_cla.CDbgSignalSnapshotNode1Eap1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap0Lo

*Table 2339. smc_cla.CDbgSignalSnapshotNode2Eap0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap1Lo

*Table 2340. smc_cla.CDbgSignalSnapshotNode2Eap1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap0Lo

*Table 2341. smc_cla.CDbgSignalSnapshotNode3Eap0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap1Lo

*Table 2342. smc_cla.CDbgSignalSnapshotNode3Eap1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgClaTimeMatch

*Table 2343. smc_cla.CDbgClaTimeMatch fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `TimeMatchVal` | RW | 0x0 | - |

### CDbgSignalMask2Lo

*Table 2344. smc_cla.CDbgSignalMask2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch2Lo

*Table 2345. smc_cla.CDbgSignalMatch2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalMask3Lo

*Table 2346. smc_cla.CDbgSignalMask3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch3Lo

*Table 2347. smc_cla.CDbgSignalMatch3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgNode0Eap2

*Table 2348. smc_cla.CDbgNode0Eap2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgNode0Eap3

*Table 2349. smc_cla.CDbgNode0Eap3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Action3` | RW | 0x0 | Select an Action |
| 57:52 | `Action2` | RW | 0x0 | Select an Action |
| 51:44 | `Udf` | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | `EventType2` | RW | 0x0 | Select a trigger event |
| 37 | `CustomAction1Enable` | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | `CustomAction0Enable` | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | `CustomAction1` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode1Eap2

*Table 2350. smc_cla.CDbgNode1Eap2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | CustomAction0 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | EventType1 | RW | 0x0 | Select a trigger event |
| 21:16 | EventType0 | RW | 0x0 | Select a trigger event |
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode1Eap3

*Table 2351. smc_cla.CDbgNode1Eap3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | CustomAction0 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | EventType1 | RW | 0x0 | Select a trigger event |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 21:16 | EventType0 | RW | 0x0 | Select a trigger event |
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode2Eap2

*Table 2352. smc_cla.CDbgNode2Eap2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | CustomAction0 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | EventType1 | RW | 0x0 | Select a trigger event |
| 21:16 | EventType0 | RW | 0x0 | Select a trigger event |
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode2Eap3

*Table 2353. smc_cla.CDbgNode2Eap3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | CustomAction0 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 27:22 | EventType1 | RW | 0x0 | Select a trigger event |
| 21:16 | EventType0 | RW | 0x0 | Select a trigger event |
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode3Eap2

*Table 2354. smc_cla.CDbgNode3Eap2 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 31:28 | CustomAction0 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | EventType1 | RW | 0x0 | Select a trigger event |
| 21:16 | EventType0 | RW | 0x0 | Select a trigger event |
| 15:14 | LogicalOp | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | Action1 | RW | 0x0 | Select an Action |
| 7:2 | Action0 | RW | 0x0 | Select an Action |
| 1:0 | DestNode | RW | 0x0 | Select Destination Node |

### CDbgNode3Eap3

*Table 2355. smc_cla.CDbgNode3Eap3 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | Action3 | RW | 0x0 | Select an Action |
| 57:52 | Action2 | RW | 0x0 | Select an Action |
| 51:44 | Udf | RW | 0x0 | User Defined Function with event type 0, 1, and 2 as arguments |
| 43:38 | EventType2 | RW | 0x0 | Select a trigger event |
| 37 | CustomAction1Enable | RW | 0x0 | custom_action1 is valid only if custom_action1_enable is set |
| 36 | CustomAction0Enable | RW | 0x0 | custom_action0 is valid only if custom_action0_enable is set |
| 35:32 | CustomAction1 | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `CustomAction0` | RW | 0x0 | Select the bit position of custom action bus to be set when EAP trigger is met. External blocks can define and implement the specific action for a given bit. |
| 27:22 | `EventType1` | RW | 0x0 | Select a trigger event |
| 21:16 | `EventType0` | RW | 0x0 | Select a trigger event |
| 15:14 | `LogicalOp` | RW | 0x0 | Relation to be satisfied among events to activate the actions |
| 13:8 | `Action1` | RW | 0x0 | Select an Action |
| 7:2 | `Action0` | RW | 0x0 | Select an Action |
| 1:0 | `DestNode` | RW | 0x0 | Select Destination Node |

### CDbgSignalSnapshotNode0Eap2Lo

*Table 2356. smc_cla.CDbgSignalSnapshotNode0Eap2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode0Eap3Lo

*Table 2357. smc_cla.CDbgSignalSnapshotNode0Eap3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap2Lo

*Table 2358. smc_cla.CDbgSignalSnapshotNode1Eap2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap3Lo

*Table 2359. smc_cla.CDbgSignalSnapshotNode1Eap3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap2Lo

*Table 2360. smc_cla.CDbgSignalSnapshotNode2Eap2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap3Lo

*Table 2361. smc_cla.CDbgSignalSnapshotNode2Eap3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap2Lo

*Table 2362. smc_cla.CDbgSignalSnapshotNode3Eap2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap3Lo

*Table 2363. smc_cla.CDbgSignalSnapshotNode3Eap3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalDelayMuxSel

*Table 2364. smc_cla.CDbgSignalDelayMuxSel fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Rsvd` | RW | 0x0 | - |
| 31:30 | `Muxselseg15` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 29:28 | `Muxselseg14` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 27:26 | `Muxselseg13` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 25:24 | `Muxselseg12` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 23:22 | `Muxselseg11` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 21:20 | `Muxselseg10` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 19:18 | `Muxselseg9` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 17:16 | `Muxselseg8` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |
| 15:14 | `Muxselseg7` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 7 |
| 13:12 | `Muxselseg6` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 6 |
| 11:10 | `Muxselseg5` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 5 |
| 9:8 | `Muxselseg4` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 4 |
| 7:6 | `Muxselseg3` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 3 |
| 5:4 | `Muxselseg2` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 2 |
| 3:2 | `Muxselseg1` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 1 |
| 1:0 | `Muxselseg0` | RW | 0x0 | Dbg Mux Shifted signals Select Bits for Lane 0 |

### CDbgClaXtriggerTimestretch

*Table 2365. smc_cla.CDbgClaXtriggerTimestretch fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:16 | `Rsvd` | RW | 0x0 | - |
| 15:8 | `Xtrigger1Stretch` | RW | 0x0 | - |
| 7:0 | `Xtrigger0Stretch` | RW | 0x0 | - |

### CDbgClaTimestamp

*Table 2366. smc_cla.CDbgClaTimestamp fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:8 | `TimestampUpper` | RW | 0x0 | Current timestamp value (upper 56 bits) |
| 7:0 | `TimestampLower` | RW | 0x0 | Current timestamp value (lower 8 bits) |

### CDbgClaTimestampSync

*Table 2367. smc_cla.CDbgClaTimestampSync fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `TimestampSync` | RW | 0x0 | Sync value to load into timestamp |

### CDbgClaTimestampConfig

*Table 2368. smc_cla.CDbgClaTimestampConfig fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:10 | `Rsvd1` | RW | 0x0 | Reserved |
| 9 | `TsCapture` | RW | 0x0 | When this bit is set, a cross trigger 0 event will cause the value from Timestamp register to be written into the TimestampCapture register. No effect in the SMC, where the CLA timestamp counts the local time tick (timestamp scheme 0). |
| 8:1 | `DebugMarker` | RW | 0x0 | Debug marker output value |
| 0 | `Resync` | RW | 0x0 | When set, timestamp is loaded from timestamp_sync on xtrigger posedge. Auto-clears after load. |

### CDbgClaTimestampOffset

*Table 2369. smc_cla.CDbgClaTimestampOffset fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Rsvd0` | RW | 0x0 | Reserved |
| 55:0 | `Offset` | RW | 0x0 | Timestamp offset value. No effect in the SMC. |

### CDbgSignalMask0Hi

*Table 2370. smc_cla.CDbgSignalMask0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch0Hi

*Table 2371. smc_cla.CDbgSignalMatch0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalMask1Hi

*Table 2372. smc_cla.CDbgSignalMask1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch1Hi

*Table 2373. smc_cla.CDbgSignalMatch1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalMask2Hi

*Table 2374. smc_cla.CDbgSignalMask2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch2Hi

*Table 2375. smc_cla.CDbgSignalMatch2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgSignalMask3Hi

*Table 2376. smc_cla.CDbgSignalMask3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Mask to be applied to debug bus before match |

### CDbgSignalMatch3Hi

*Table 2377. smc_cla.CDbgSignalMatch3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value used for debug bus match. |

### CDbgMuxSelHi

*Table 2378. smc_cla.CDbgMuxSelHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Muxselseg15` | RW | 0x0 | No effect in the SMC. |
| 57:52 | `Muxselseg14` | RW | 0x0 | No effect in the SMC. |
| 51:46 | `Muxselseg13` | RW | 0x0 | No effect in the SMC. |
| 45:40 | `Muxselseg12` | RW | 0x0 | No effect in the SMC. |
| 39:34 | `Muxselseg11` | RW | 0x0 | No effect in the SMC. |
| 33:28 | `Muxselseg10` | RW | 0x0 | No effect in the SMC. |
| 27:22 | `Muxselseg9` | RW | 0x0 | No effect in the SMC. |
| 21:16 | `Muxselseg8` | RW | 0x0 | No effect in the SMC. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 15:8 | `Rsvd158` | RW | 0x0 | - |
| 7:2 | `Dbmid` | RW | 0x0 | No effect in the SMC. |
| 1:0 | `Dbmmode` | RW | 0x0 | No effect in the SMC. |

### CDbgTransitionMaskHi

*Table 2379. smc_cla.CDbgTransitionMaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgTransitionFromValueHi

*Table 2380. smc_cla.CDbgTransitionFromValueHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgTransitionToValueHi

*Table 2381. smc_cla.CDbgTransitionToValueHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgOnesCountMaskHi

*Table 2382. smc_cla.CDbgOnesCountMaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | - |

### CDbgAnyChangeHi

*Table 2383. smc_cla.CDbgAnyChangeHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Mask` | RW | 0x0 | - |

### CDbgSignalSnapshotNode0Eap0Hi

*Table 2384. smc_cla.CDbgSignalSnapshotNode0Eap0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode0Eap1Hi

*Table 2385. smc_cla.CDbgSignalSnapshotNode0Eap1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap0Hi

*Table 2386. smc_cla.CDbgSignalSnapshotNode1Eap0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap1Hi

*Table 2387. smc_cla.CDbgSignalSnapshotNode1Eap1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap0Hi

*Table 2388. smc_cla.CDbgSignalSnapshotNode2Eap0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap1Hi

*Table 2389. smc_cla.CDbgSignalSnapshotNode2Eap1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap0Hi

*Table 2390. smc_cla.CDbgSignalSnapshotNode3Eap0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap1Hi

*Table 2391. smc_cla.CDbgSignalSnapshotNode3Eap1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode0Eap2Hi

*Table 2392. smc_cla.CDbgSignalSnapshotNode0Eap2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode0Eap3Hi

*Table 2393. smc_cla.CDbgSignalSnapshotNode0Eap3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap2Hi

*Table 2394. smc_cla.CDbgSignalSnapshotNode1Eap2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode1Eap3Hi

*Table 2395. smc_cla.CDbgSignalSnapshotNode1Eap3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap2Hi

*Table 2396. smc_cla.CDbgSignalSnapshotNode2Eap2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode2Eap3Hi

*Table 2397. smc_cla.CDbgSignalSnapshotNode2Eap3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap2Hi

*Table 2398. smc_cla.CDbgSignalSnapshotNode3Eap2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgSignalSnapshotNode3Eap3Hi

*Table 2399. smc_cla.CDbgSignalSnapshotNode3Eap3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | R | 0x0 | - |

### CDbgLfsr

*Table 2400. smc_cla.CDbgLfsr fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Lfsr` | RW | 0x0 | Value of the LFSR register. |
| 0 | `LfsrActive` | RW | 0x0 | Enable the LFSR register. |

### CDbgLfsrMask

*Table 2401. smc_cla.CDbgLfsrMask fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:1 | `Mask` | RW | 0x0 | Value to mask the LFSR register. |
| 0 | `Rsvd0` | RW | 0x0 | Reserved. |

### CDbgTimestampCapture

*Table 2402. smc_cla.CDbgTimestampCapture fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Timestamp` | R | 0x0 | Timestamp capture value. |

### CDbgCompare0Lo

*Table 2403. smc_cla.CDbgCompare0Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare0MaskLo

*Table 2404. smc_cla.CDbgCompare0MaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare1Lo

*Table 2405. smc_cla.CDbgCompare1Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare1MaskLo

*Table 2406. smc_cla.CDbgCompare1MaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare2Lo

*Table 2407. smc_cla.CDbgCompare2Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare2MaskLo

*Table 2408. smc_cla.CDbgCompare2MaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare3Lo

*Table 2409. smc_cla.CDbgCompare3Lo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare3MaskLo

*Table 2410. smc_cla.CDbgCompare3MaskLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare0Hi

*Table 2411. smc_cla.CDbgCompare0Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare0MaskHi

*Table 2412. smc_cla.CDbgCompare0MaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare1Hi

*Table 2413. smc_cla.CDbgCompare1Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare1MaskHi

*Table 2414. smc_cla.CDbgCompare1MaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare2Hi

*Table 2415. smc_cla.CDbgCompare2Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare2MaskHi

*Table 2416. smc_cla.CDbgCompare2MaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### CDbgCompare3Hi

*Table 2417. smc_cla.CDbgCompare3Hi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value for arithmetic comparison. |

### CDbgCompare3MaskHi

*Table 2418. smc_cla.CDbgCompare3MaskHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Value` | RW | 0x0 | Value to mask the arithmetic comparison. |

### Scratch

*Table 2419. smc_cla.Scratch fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:0 | `Data` | RW | 0x0 | - |

### Trdstcontrol

*Table 2420. smc_cla.Trdstcontrol fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:27 | `Reserved` | - | - | Reserved |
| 26:24 | `Trdstformat` | RW | 0x3 | bit[0]: XOR Enable, bit[1]: VLT Enable. Supported values : 2’b3 (XOR+VLT Compression), 2’b1 (XOR Compresion), 2’b0 (No Compression) |
| 23:20 | `Trdstsyncmax` | RW | 0x0 | When trDstSyncMode is set to 2’b10, timestamp will be sent for every 2^(trDstSyncMax + 4) Cluster clocks. |
| 19:18 | `Reserved` | - | - | Reserved |
| 17:16 | `Trdstsyncmode` | RW | 0x0 | Periodic sync mode. 2’b00: off. 2’b01: stream counting, with trDstVendorStreamLength frames between non-compressed packets. 2’b10: send a timestamp packet every 2^(trDstSyncMax + 4) Cluster clocks. 2’b11: no effect, as 2’b00. |
| 15 | `Trdstinhibitsrc` | R | 0x0 | - |
| 14 | `Reserved` | - | - | Reserved |
| 13 | `Trdstinststallena` | R | 0x0 | - |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 12 | `Trdstinststalloroverflow` | R | 0x0 | - |
| 11 | `Trdstinsttriggerenable` | RW | 0x0 | - |
| 10 | `Reserved` | - | - | Reserved |
| 9 | `Trdstcontext` | R | 0x0 | - |
| 8:7 | `Reserved` | - | - | Reserved |
| 6:4 | `Trdstinstmode` | R | 0x6 | - |
| 3 | `Trdstempty` | R | 0x1 | - |
| 2 | `Trdstinsttracing` | RW | 0x0 | - |
| 1 | `Trdstenable` | RW | 0x0 | - |
| 0 | `Trdstactive` | RW | 0x0 | - |

### Trdstimpl

*Table 2421. smc_cla.Trdstimpl fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Trdsttimestampconfig` | RW | 0x0 | DST timestamp configuration. 0: timestamp input form external source (CoreTime). 1: timestamp from CLA timesync. |
| 30:28 | `Trdstvendorstreamlength` | RW | 0x4 | Specify Stream length. A stream starts with "no compressed packet". Stream length specifies number of frames before a non-compressed packet is sent, and applies only when trDstSyncMode is 2'b01. Values 0 to 4 give Stream Length = 32* 2^trDstVendorStreamLength (32 to 512 frames); values 5 to 7 give 32. |
| 27:24 | `Trdstvendorframelength` | RW | 0x1 | Specify frame length. Frame Length = trDstVendorFrameLength* 64 bytes; Frame Length should be a multiple of Bank Data Width. Only 1, 2, 4 and 8 (64, 128, 256 and 512 bytes) are supported, because the DST sink finds frame boundaries at the power of two given by the highest set bit of its frame length. Program the sink's trDstRamVendorFrameLength with the same value. |
| 23:20 | `Trdstprotocolminor` | R | 0x0 | - |
| 19:16 | `Trdstprotocolmajor` | R | 0x1 | - |
| 15:12 | `Reserved` | - | - | Reserved |
| 11:8 | `Trdstcomptype` | R | 0x1 | - |
| 7:4 | `Trdstverminor` | R | 0x0 | - |
| 3:0 | `Trdstvermajor` | R | 0x1 | - |

### Trdstinstfeatures

*Table 2422. smc_cla.Trdstinstfeatures fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | `Trdstsrcbits` | RW | 0x4 | - |
| 27:16 | `Trdstsrcid` | RW | 0x0 | Source ID sent in the trace packet headers. Only bits [3:0] are used. |
| 15:9 | `Reserved` | - | - | Reserved |
| 8 | `Trdstinstenrepeatedhistory` | R | 0x0 | - |
| 7:2 | `Reserved` | - | - | Reserved |
| 1 | `Trdstinstnotrapaddr` | R | 0x0 | - |
| 0 | `Trdstinstnoaddrdiff` | R | 0x0 | - |

#### CDbgDebugTraceCfg

*Table 2423. smc_cla.CDbgDebugTraceCfg fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:22 | `Reserved` | - | - | Reserved |
| 21 | `FrameClosureMode` | RW | 0x0 | Closure Mode: 1: Close frame with a packet no larger than max packet size before any overflow (due to packets crossing frame boundary). 0: Close frame when packets cross boundary. Push back packet crossing frame boundary to packet generator. |
| 20 | `FrameModeEnable` | RW | 0x1 | Enable Frame Mode (Frame Mode: Always pack "frame length" number of bytes for trace transmission). If flush to memory is stalled (due to silicon bug), we can have incomplete packets in memory. Frame Mode will help SW decoder to avoid reading incomplete trasnmitted packets. In Frame Mode, SW will get a memory pointer that is aligned to Frame Length. HW will ensure there are no incomplete packets incluced in the region pointed by memory pointer. |
| 19:16 | `Reserved` | - | - | Reserved |
| 15:12 | `FrameLenghtInBytes` | RW | 0x2 | Use trDstImpl[trDstVendorFrameLength] for Frame Length programming. Register field ZBB-ed. |
| 11:4 | `TraceFrameFillByte` | RW | 0x81 | Use "Filler Packet" to align a stream of trace packets within a frame boundary. |
| 3:0 | `TraceSourceId` | RW | 0x0 | Stored but not used. The trace packet headers carry trDstInstFeatures[trDstSrcID]. |

#### dst.ScratchLo

*Table 2424. smc_cla.dst.ScratchLo fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

#### dst.ScratchHi

*Table 2425. smc_cla.dst.ScratchHi fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `Data` | RW | 0x0 | - |

### 2.1.6.9. eFuse Shadow Register Scan Protection

DFT scan test and debug scandump share the scan chain infrastructure. Any flop placed on a scan chain can be shifted out, and any flop on a dumpable chain can be observed by a debug user. The SMC eFuse shadow registers hold the `LOCKS` field, which is the sole enforcement point for read and write access control over the entire SMC fuse map. If those flops were scannable, an attacker could shift a cleared `LOCKS` value into the shadow array and defeat the access controls without touching the fuse macro itself. This section describes how the `LOCKS` shadow words are excluded from every scan chain.

The asset classes referenced here (Class 1 / Class 2 / Class 3) are defined authoritatively in the SEP Lifecycle Controller chapter, *Asset and Scan-Chain Protection*. This section records how they apply to the SMC eFuse controller.

#### SMC eFuse Asset Classification

Table 2426 relates the SMC fuse contents to their security class and scan policy.

*Table 2426. SMC eFuse shadow register classification*

| Class | SMC content | Scan policy |
|---|---|---|
| Class 1a (confidentiality) | *none* | The SMC fuse map holds no device secrets — it carries identifiers (`JTAG_PUBLIC_IDENTITY`, `I2C_I3C_ID`), boot configuration (`SMC_CONFIG`, `OCCP_TRANSPORT_TIMEOUT`), and spare regions. There are no keys, no UIDs used as key material, and no token registers. Consequently no confidentiality masking is required. |
| Class 1b (integrity) | `LOCKS` | Excluded from every scan chain. Never scannable, never dumpable, for any LC state or debug grant. |
| Class 2 (secret-bearing) | *none* | The SMC eFuse controller instantiates no cryptographic engine and no key vault, so it contributes no Class 2 scan roots and needs no dedicated secret-bearing chain. |
| Class 3 (general) | All remaining fuse-map words | Normal scan and scandump handling, subject to the SMC debug policy. |

Because SMC has no Class 1a asset, `secure_tm` is not used for secret disconnection in the SMC eFuse controller: `secure_tm_i` is tied to `1'b0` and `SECRET_SHADOW_RANGES` is left at `'0`. This differs from SEP, where `secure_tm` disconnects the UID / `CLASS_KEY` sources from their Class 2 consumers.

### Implementation

`efuse_shadow_regs` keeps its shadow storage in two separately named arrays and reconstructs the public register view from both. The array holding Class 1 words is named with an `_n0_scan` suffix, which is the reserved naming handle the synthesis scan-insertion flow uses to identify the flops it must leave off all scan chains:

- `shadow_efuse_values_n0_scan` — Class 1 words, excluded from scan.
- `shadow_efuse_values` — Class 3 words, normal scan handling.

Which words land in the Class 1 array is set by the `CLASS1_SHADOW_RANGES` parameter. `smc_efuse_wrapper` drives it from `smc_efuse_pkg::Class1ShadowRanges`, which derives the range from the generated RDL metadata rather than hard-coding word indices, so the exclusion follows the register map if the map is regenerated:

```
localparam efuse_pkg::shadow_word_range_map_t Class1ShadowRanges = '{
    0: efuse_pkg::make_shadow_word_range(
        LOCKS_REG_OFFSET, $bits(smc_efuse_map_locks_reg_t), NumFuseWordWidth),
    default: '0
};
```

`LOCKS` sits at fuse-map offset `0x0` and is 64 bits wide, so this resolves to shadow words 0 and 1.

Table 2427 maps fuse fields to their shadow-register word ranges.

*Table 2427. SMC shadow word allocation*

| Quantity | Value |
|---|---|
| Fuse map size | 1 KB (8192 bits, 256 x 32-bit words) |
| Class 1 words (`shadow_efuse_values_n0_scan`) | 2 (words 0-1, `LOCKS`) |
| Class 3 words (`shadow_efuse_values`) | 254 |

Three `ASSERT_INIT` checks in `efuse_shadow_regs` verify at elaboration that the configured ranges are well formed, non-overlapping, and inside the shadow array, so a bad range map is a build failure rather than a silent loss of protection.

#### Downstream Lock Path

Excluding the `LOCKS` shadow flops is only sufficient if no scannable flop downstream of them re-derives the lock decision. In the SMC eFuse controller this holds by construction: `efuse_shadow_reg_access_control`, which compares the incoming APB address against the field map and produces the `write_locked` / `read_locked` decisions, is purely combinational. There is no intermediate state element between the Class 1 shadow words and the APB write-enable and read-data paths that they gate.

The one sequential consumer of that logic is `locked_field_access_interrupt_o`, which reports that a locked-field access was attempted. It carries an access-violation event, not lock state, so observing it does not reveal or alter the `LOCKS` value.

#### Notes for DFT

- The `_n0_scan` suffix is a Class 1 marker and takes precedence over any enclosing scan hierarchy. Do not rename it; the exclusion is carried entirely by the signal name, as there is no separate constraint list in this repository.
- `efuse_shadow_regs` also declares `chiplet_state_change_completed_n0_scan` and `sop_state_change_completed_n0_scan`. These serve the SEP RMA token / LC state transition path. In the SMC instance `HAS_LC_STATE` is `1'b0` and both are driven to a constant `1'b1`, so they hold no SMC state; they are expected to be constant-propagated away.
- Adding a secret or a new integrity-critical field to the SMC fuse map requires extending `Class1ShadowRanges` (and, for a confidentiality asset, `SECRET_SHADOW_RANGES` plus a real `secure_tm_i` source). Adding the register to the RDL alone does not confer scan protection.

#### Verification

`smc_efuse_shadow_lock_matrix_test` exercises reads and writes across the `LOCKS` matrix and therefore covers the relocated words functionally, confirming that moving `LOCKS` into the Class 1 array does not change access-control behaviour.

### 2.1.6.10. Memory Map and Register Reference

#### Memory Map

This section provides a comprehensive overview of the SMC memory map. The SMC address space is organized to provide access to various functional units including CPU control, peripheral interfaces, system management functions, and fabric control mechanisms.

All addresses below are expressed as `BASE + offset`, where `BASE` can be either the local alias (`LOCAL_BASE`, fixed read-only at `0xC000_0000`) or the global system address (`GLOBAL_BASE`, programmable by firmware). The fabric routes transactions to SMC resources identically regardless of which base is used, as long as the access falls inside the relevant local or global aperture. Both apertures are sized by the SMC `REGION_SIZE` CSR, which resets to 16 MiB and may be reprogrammed by firmware. Address ranges include both endpoints. Addresses are 32-bit aligned and follow standard AXI4-Lite addressing conventions. For details on how `LOCAL_BASE` and `GLOBAL_BASE` relate and how an adopter maps this aperture into the system address space, refer to the SMC Address Map Integration section of the Integration Guide.

#### Address Space Organization

The generated component map reports each allocated aperture separately from the extent occupied by registers.

*Table 2428. SMC Functional Address-Space Organization*

| Address Range | Size | Unit | Description |
|---|---|---|---|
| BASE + 0x0 – BASE + 0xFFF | 4 KiB | Watchdog Timers | Per-core watchdog timers. |

| Address Range | Size | Unit | Description |
|---|---|---|---|
| BASE + 0x2000 – BASE + 0xBFFF | 40 KiB | System and Peripheral Control | Reset, GPIO, serial peripherals, eFuse, telemetry, timer, DTP, and DFT status. |
| BASE + 0x10000 – BASE + 0x16FFF | 28 KiB | Fabric Control | Base configuration, address remapping, and traffic filters. |
| BASE + 0x18000 – BASE + 0x37FFF | 128 KiB | Mailbox | Thirty-two bidirectional mailbox pairs. |
| BASE + 0x38000 – BASE + 0x382FF | 768 B | Data Processing | DMA controller and memory zeroer. |
| BASE + 0x39000 – BASE + 0x3FFFF | 28 KiB | CPU and I/O Control | CPU-cluster control and OCA I3C controller. |
| BASE + 0x40000 – BASE + 0x15FFFF | 1152 KiB | Local Memories | Scratchpad ROM and RAM. |
| BASE + 0x160000 – BASE + 0x163FFF | 16 KiB | CLA | Core Logic Analyzer debug and trace. |
| BASE + 0x400000 – BASE + 0x406547 | 25928 B | Adopter External | Adopter-owned technology-specific and supplementary peripherals. |
| BASE + 0x1000000 – BASE + 0x1FFFFFF | 16 MiB | Address Remapping | Machine-mode and hypervisor remapped regions. |
| BASE + 0x4000000 – BASE + 0x7FFFFFF | 64 MiB | PLIC | Platform-Level Interrupt Controller aperture. |
| BASE + 0x8000000 – BASE + 0x8013FFF | 80 KiB | Core-Local Timer and Errors | Core-local timer and per-core bus-error units. |

*Table 2429. SMC Component Address Map*

| Address Range | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|
| BASE + 0x0 – BASE + 0x3FF | 1 KiB | 36 B | Watchdog Timer Address Map | 1 instance | - | Core 0 watchdog timer. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x400 – BASE + 0x7FF | 1 KiB | 36 B | Watchdog Timer Address Map | 1 instance | - | Core 1 watchdog timer. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x800 – BASE + 0xBFF | 1 KiB | 36 B | Watchdog Timer Address Map | 1 instance | - | Core 2 watchdog timer. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0xC00 – BASE + 0xFFF | 1 KiB | 36 B | Watchdog Timer Address Map | 1 instance | - | Core 3 watchdog timer. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|
| BASE + 0x1000 – BASE + 0x1867 | 2152 B | 2152 B | Debug Module Hart Interface | 1 instance | - | Memory region through which a hart in Debug Mode exchanges state with the Debug Module: handshake mailboxes, generated instructions, program buffer, data registers, per-hart flags and the debug ROM. Only the harts of the CPU cluster reach this region; SMC bus initiators outside the cluster do not. The debugger-facing Debug Module registers (dmcontrol, dmstatus, abstractcs, sbcs, …) are reached over the Debug Module Interface and do not appear in this map. | DECERR, 0xBADCAB1E / DECERR[1] | - |
| BASE + 0x2000 – BASE + 0x27FF | 2 KiB | 204 B | Reset Unit Address Map | 1 instance | - | System reset control. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x2800 – BASE + 0x2FFF | 2 KiB | 524 B | SMC Miscellaneous Wrapper Address Map | 1 instance | - | Aggregates the SMC scratch, chip-configuration, and NDM-reset register blocks. | DECERR, 0xBADCAB1E / SLVERR | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x3000 – BASE + 0x3FFF | 4 KiB | 1040 B | GPIO Interface Address Map | 65 instances | 16 B | GPIO interface configuration: pad/core routing, receive and transmit enables, and interrupt control. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|
| BASE + 0x4000 – BASE + 0x4FFF | 4 KiB | 92 B | AVSBus Controller Address Map | 1 instance | - | Command, readback, status, and interrupt registers for the AVSBus (V1.3.1) controller. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x5000 – BASE + 0x5FFF | 4 KiB | 3596 B | I2C Wrapper Address Map | 1 instance | - | I2C and SMBus controller. | DECERR, 0xBADCAB1E / SLVERR | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x6000 – BASE + 0x6FFF | 4 KiB | 4 KiB | UART Wrapper Address Map | 1 instance | - | UART controller. | OKAY, 0x0 / OKAY; between sub-blocks: DECERR, 0xBADCAB1E / SLVERR | - |
| BASE + 0x7000 – BASE + 0x7FFF | 4 KiB | 1 KiB | Efuse address map of physical efuse macro | 1 instance | - | Address map for efuse shadow registers providing access to various chip configuration and ID fields | - | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x8000 – BASE + 0x8FFF | 4 KiB | 28 B | eFuse Interface Control Address Map | 1 instance | - | eFuse command and status registers. | - | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x9000 – BASE + 0x9FFF | 4 KiB | 768 B | Telemetry Receiver Wrapper Address Map | 1 instance | - | Telemetry data receiver. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0xA000 – BASE + 0xAFFF | 4 KiB | 36 B | OCTS System Timer Address Map | 1 instance | - | Open Chiplet Time Synchronizatio n (OCTS) System Timer Registers | - | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0xB000 – BASE + 0xB7FF | 2 KiB | 2 KiB | dtp_ctrl_reg | 1 instance | - | 512-entry DTP control block. | Forwarded | - |
| BASE + 0xB800 – BASE + 0xBFFF | 2 KiB | 24 B | DFX Control and Status Address Map | 1 instance | - | DFT status exposed to boot firmware. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x10000 – BASE + 0x11FFF | 8 KiB | 76 B | SMC Base Configuration Address Map | 1 instance | - | SMC base-address and aperture configuration. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x12000 – BASE + 0x12FFF | 4 KiB | 256 B | AXI Alias Remap Address Map | 8 instances | 32 B | TLB Registers for address remapping and attribute tagging | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|
| BASE + 0x13000 – BASE + 0x13FFF | 4 KiB | 64 B | Output Remap Address Map | 8 instances | 8 B | Address-translation registers that remap outbound transactions and tag their attributes. | - | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x14000 – BASE + 0x14FFF | 4 KiB | 64 B | Output Remap Address Map | 8 instances | 8 B | Address-translation registers that remap outbound transactions and tag their attributes. | - | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x15000 – BASE + 0x15FFF | 4 KiB | 512 B | AXI Filter Control Address Map | 16 instances | 32 B | Per-filter access-control registers defining permitted address ranges and transaction attributes. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x16000 – BASE + 0x16FFF | 4 KiB | 512 B | AXI Filter Control Address Map | 16 instances | 32 B | Per-filter access-control registers defining permitted address ranges and transaction attributes. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / SLVERR |
| BASE + 0x18000 – BASE + 0x37FFF | 128 KiB | 129104 B | SMC AXI-Lite Mailbox Wrapper Address Map | 1 instance | - | Thirty-two bidirectional mailbox pairs. | SLVERR, 0x0 / SLVERR | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x38000 – BASE + 0x381FF | 512 B | 312 B | iDMA Controller Address Map | 1 instance | - | Direct-memory-access controller. | OKAY, 0xFFFFFFFF / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x38200 – BASE + 0x382FF | 256 B | 24 B | Zeroer Control Address Map | 1 instance | - | Control and status registers for the AXI memory-zeroing engine. | - | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x39000 – BASE + 0x39FFF | 4 KiB | 704 B | CPU Control Address Map | 1 instance | - | SMC CPU-cluster control. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x3A000 – BASE + 0x3FFFF | 24 KiB | 24 KiB | OCA I3C Controller Wrapper Address Map | 1 instance | - | Wrapper address map instantiating the per-controller I3C CSR blocks. | OKAY, 0x0 / OKAY[2] | - |

| Address Range | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|
| BASE + 0x40000 – BASE + 0x5FFFF | 128 KiB | 128 KiB | spm_rom_memory | 1 instance | - | Scratchpad ROM memory, present only in non-CPL configurations. | -[3] | - |
| BASE + 0x60000 – BASE + 0x15FFFF | 1 MiB | 1 MiB | spm_memory | 1 instance | - | Scratch RAM Memory - only present for non-CPL configurations | - | - |
| BASE + 0x160000 – BASE + 0x163FFF | 16 KiB | 16 KiB | SMC DFD MMR Aperture Address Map | 1 instance | - | Core Logic Analyzer debug and trace aperture. | OKAY, 0x0 / OKAY | - |
| BASE + 0x400000 – BASE + 0x406547 | 25928 B | 25928 B | SMC External Window Address Map | 1 instance | - | Adopter-owned external peripheral aperture. | Adopter-defined | - |
| BASE + 0x1000000 – BASE + 0x17FFFFF | 8 MiB | 8 MiB | mmode_region | 1 instance | - | Address region whose accesses an output remap redirects. | Forwarded | - |
| BASE + 0x1800000 – BASE + 0x1FFFFFF | 8 MiB | 8 MiB | xvisor_region | 1 instance | - | Address region whose accesses an output remap redirects. | Forwarded | - |
| BASE + 0x4000000 – BASE + 0x7FFFFFF | 64 MiB | 2125832 B | PLIC Address Map | 1 instance | - | Platform-Level Interrupt Controller. | OKAY, 0x0 / OKAY[4] | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8000000 – BASE + 0x800FFFF | 64 KiB | 48 KiB | CLINT Address Map | 1 instance | - | Core-Local Interruptor. | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8010000 – BASE + 0x8010FFF | 4 KiB | 48 B | Bus Error Unit Address Map | 1 instance | - | Core 0 bus-error unit. | - | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8011000 – BASE + 0x8011FFF | 4 KiB | 48 B | Bus Error Unit Address Map | 1 instance | - | Core 1 bus-error unit. | - | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8012000 – BASE + 0x8012FFF | 4 KiB | 48 B | Bus Error Unit Address Map | 1 instance | - | Core 2 bus-error unit. | - | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8013000 – BASE + 0x8013FFF | 4 KiB | 48 B | Bus Error Unit Address Map | 1 instance | - | Core 3 bus-error unit. | - | DECERR, 0xBADCAB1E / DECERR |
| BASE + 0x8014000 – BASE + 0x8014FFF | 4 KiB | 4 KiB | smc_cluster_error_device | 1 instance | - | Error responder for unmapped cluster-bus accesses. | DECERR, 0xBADCAB1E / DECERR[1] | - |

*Notes*

[1] Only the harts of the CPU cluster reach this unit. An access from outside the cluster answers the same at every address in its aperture.
[2] Each instance implements 16 DAT and 16 DCT entries: the DAT at 0x400-0x7FF repeats every 0x80 bytes, the DCT at 0x800-0xFFF repeats every 0x100 bytes, and DCT writes are dropped.
[3] A write answers SLVERR and leaves the ROM unchanged.
[4] 0x544, 0x102C and 0x202C are the upper halves of live 64-bit words, so a 64-bit read there returns live data.

The last two columns state how the SMC answers a 32-bit access that no register or memory backs, as seen at its inbound AXI port, and Decode Response Codes defines them and their codes.

An address between unit apertures reaches no unit. A read there answers `DECERR` with data `0xBADCAB1E`, and a write answers `DECERR`, except from `BASE + 0x001_7000` through `BASE + 0x001_7FFF`, where it answers `SLVERR`.

An address inside neither the local nor the global aperture answers `DECERR` to reads and writes. The M-mode and Xvisor remap regions forward accesses from SMC initiators only; an access to them from the inbound port answers `DECERR`.

The inbound port reaches a unit only while its whole aperture lies below `BASE + REGION_SIZE`. At the reset value the PLIC, CLINT and bus-error units lie beyond it and answer `DECERR`. The SMC harts reach the PLIC and CLINT inside the CPU cluster regardless of `REGION_SIZE`.

The I2C, UART, I3C, AVSBus and Telemetry Receiver windows answer `SLVERR`, with read data `0xBADCAB1E`, while the peripheral's `CLOCK_GATE_CONTROL` enable in SMC_BASE_CONFIG is set and its clock is stopped.

**AXI-Lite External Window**

The `smc_external` window at `BASE + 0x040_0000` is the adopter extension window. OCAH exposes this AXI-Lite bus so an adopter can attach their own technology- and vendor-specific implementations of memory-mapped blocks. The window is split into two regions (see `smc_external.rdl`). Apart from the eFuse SHIM control block described below, hardware treats both identically; the split is purely organizational:

- **Mandatory** (`BASE + 0x040_0000`): blocks every integration must implement. Their function is dictated by the OCAH specifications, but their implementation is inherently technology- and vendor-specific — the eFuse SHIM, the per-pad GPIO controls, the PLL wrapper and the strap registers at `BASE + 0x040_3000`.
- **Supplementary** (`BASE + 0x040_4000`): any additional memory-mapped devices the adopter wants to expose, such as peripheral controllers, extra pad controls and the PVT wrapper.

The generated SMC map reports this as one opaque adopter-owned aperture. Concrete placements inside it are integration-specific and are documented by the adopter. The strap registers are read-only values consumed by boot firmware; how the adopter captures them is implementation-defined. Both boot ROMs read them at `BASE + 0x040_3000`, so an integration places them there.

The eFuse SHIM control block is the exception. Its placement is fixed at the base of the window, `BASE + 0x040_0000`, and its size is the `EFUSE_SHIM_SIZE` parameter (`CFG.SMC_EFUSE_SHIM_SIZE` at the SMU). The peripheral crossbar routes that range to the eFuse controller, which forwards it unchanged on `efuse_bank_ctrl` to the adopter's SHIM. It never appears on the external port. The adopter defines the SHIM's control registers at offset `0x0` of its own `smc_external` map.

The supplementary (optional) region of the window is passed through to the SMU's `smc_external_req_o` / `smc_external_resp_i` ports for adopter integration. The adopter may implement any memory-mapped functions there, or may instead terminate the region with an error response. The reference integration (`hw/top/smc_ip_integration.sv`) terminates unclaimed accesses in this region with a DECERR response via the `prim_axi_lite_err_slv` helper; this is one integration choice, not a requirement.

### Register Interface Standards

All SMC components use standard AMBA interfaces. We summarize the interfaces in Table 2430:

*Table 2430. SMC Register Interface Standards*

| Interface | Address Width | Data Width | Description |
|---|---|---|---|
| AXI4-Lite | 32-bit | 64-bit | High-performance components |
| APB4 | 32-bit | 32-bit | Peripheral components |
| **Address Alignment:** 32-bit registers on 4-byte boundaries, 64-bit on 8-byte boundaries | | | |
| **Endianness:** Little-endian byte ordering throughout | | | |
| **Protection:** AXI protection bits used for secure/non-secure access control | | | |

### Access Control and Security

Table 2431 summarises the SMC access-control mechanisms.

*Table 2431. SMC Access Control Mechanisms*

| Mechanism | Description |
|---|---|
| Address Filtering | Configurable inbound and outbound address filters |
| Protection Levels | AXI protection attribute enforcement |
| Secure Regions | eFuse and security-critical registers require elevated privileges |
| Error Handling | Bus error units provide comprehensive error logging and reporting |

### Detailed Register Map

### External Window Reference Maps

The PLL and PVT maps below describe placeholder memory footprints used by the open verification models. Their memory words do not specify the control registers of an adopter's clock or sensor implementation. The straps map describes captured GPIO values. Placement of these blocks within the external window is integration-specific.

### PLL Wrapper Placeholder Address Map

Adopter clock-control placeholder reserving the PLL map slot; no CSRs are modeled.

**NOTE**

**Register Arrays:**

- **mem_array[954]**: 954 registers, base 0x0, stride 0x4

*Table 2432. PLL Wrapper Placeholder Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xEE4 | mem_array[954] | R | - |

### mem_array[954]

*Table 2433. pll_wrap.mem_array[954] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | - |

### PVT Wrapper Placeholder Address Map

Adopter PVT and monitor placeholder reserving the map slot; no CSRs are modeled.

**NOTE**

**Register Arrays:**

- **mem_array[594]**: 594 registers, base 0x0, stride 0x4

*Table 2434. PVT Wrapper Placeholder Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x944 | mem_array[594] | R | - |

### mem_array[594]

*Table 2435. pvt_wrap.mem_array[594] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | R | - | - |

### Straps Address Map

Configuration strap values. How they are captured is adopter-defined.

*Table 2436. Straps Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | STRAPS_LO | R | - |
| 0x4 | STRAPS_HI | R | - |

### STRAPS_LO

*Table 2437. straps.STRAPS_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `straps` | R | 0x0 | Configuration straps 0 - 31. Bits 0 - 10, 16, 17, 24 and 27 - 31 have no strap assigned; no strap function consumes them. Strap assignments in this range: * bit 11: CHIP_ID[3] - CHIP_ID is the I2C/I3C static address strap * bit 12: CHIP_ID[2] - CHIP_ID is the I2C/I3C static address strap * bit 13: BYPASS_SRAM_REPAIR - Bypass the memory repair process * bit 14: TEST_EN - Enable DFT test mode * bit 15: CHIP_ID[1] - CHIP_ID is the I2C/I3C static address strap * bit 18: BOOT_I2C - Default alternate boot channel is I3C0 and can be reconfigured to use I2C0 * bit 19: BOOT_RECOVERY - Use I3C0 for BL1 boot image loading instead of the SPI channel * bit 20: BL0_PLLCLK - Use the PLL generated clock by default in BL0, and override to use the reference clock when this strap is asserted * bit 21: STATUS_RPT_DISABLE - Disable BL0 status reporting. It is on by default * bit 22: SPI_USE_FUSED_CONFIG - Use the SPI configuration values stored in the fuse field * bit 23: CHIP_ID[0] - CHIP_ID is the I2C/I3C static address strap * bit 25: PRIMARY_CHIPLET - Identifies the primary chiplet, the boot master * bit 26: SRAM_AUTO_ZERO_DISABLE - Disable auto zero of the SMC SRAM when asserted |

#### STRAPS_HI

*Table 2438. straps.STRAPS_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:29 | `Reserved` | - | - | Reserved |
| 28:0 | `straps` | R | 0x0 | Configuration straps 32 - 60. All bits other than those listed below have no strap assigned; no strap function consumes them. Strap assignments in this range: * bit 22: MBIST_BYPASS - Ignore results of Power-On MBIST for boot * bit 26: ROTATE_UPDATE - Swaps primary and backup image in SEP BL0 ROM |

### Spare SMC Register Blocks

The following shows the detailed register map of a set of programmer-visible spare SMC blocks. They are spare but essential: for example, the scratch block provides general-purpose registers, and the chip-config and NDM-reset blocks hold system-level configuration and reset control.

### Chip Config, Scratch, NDM Reset, and Misc Wrap

### Chip Configuration Address Map

Chip version, identification, and lifecycle-state registers.

*Table 2439. Chip Configuration Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | VERSION_LO | R | - |
| 0x4 | VERSION_HI | R | - |
| 0x8 | CHIP_ID | R | - |
| 0xC | LC_STATE | R | - |

#### VERSION_LO

*Table 2440. chip_config.VERSION_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `version_lo` | R | 0x100A0 | SMC version, lower 32 bits |

#### VERSION_HI

*Table 2441. chip_config.VERSION_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `version_hi` | R | 0x0 | SMC version, upper 32 bits |

#### CHIP_ID

*Table 2442. chip_config.CHIP_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `chip_id` | R | 0x0 | Chip ID (e.g. 0 = Memory, 1 = CPU, 2 = AI Compute, 3 = Ethernet) |

#### LC_STATE

*Table 2443. chip_config.LC_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `lc_state` | R | 0xF | Lifecycle state value from SEP. Default to invalid state |

### Scratch Address Map

General-purpose scratch register bank.

> **NOTE**
>
> **Register Arrays:**
>
> - **SCRATCH[8]**: 8 registers, base 0x0, stride 0x4

*Table 2444. Scratch Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | SCRATCH[8] | RW | - |

#### SCRATCH[8]

*Table 2445. scratch.SCRATCH[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x0 | Scratch register |

### NDM Reset Control Address Map

Per-cluster NDM reset handshake with an external CPU cluster debug module

*Table 2446. NDM Reset Control Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | NDMRESET_REQUEST | R | - |
| 0x4 | NDMRESET_PROCESS | RW | - |
| 0x8 | NDMRESET_CLUSTER_COUNT | R | - |

#### NDMRESET_REQUEST

*Table 2447. ndm_reset.NDMRESET_REQUEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `ndmreset_request` | R | 0x0 | Bit i is the NDM reset request from external CPU cluster i. Bits below ndmreset_cluster_count follow the request input. Bits at and above that count read as 0. The field holds up to 32 clusters. |

#### NDMRESET_PROCESS

*Table 2448. ndm_reset.NDMRESET_PROCESS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `ndmreset_process` | RW | 0x0 | Bit i is the firmware response to the NDM reset request from external CPU cluster i. Bits below ndmreset_cluster_count are driven on the process output. Bits at and above that count are stored and are not driven. The field holds up to 32 clusters. |

### NDMRESET_CLUSTER_COUNT

*Table 2449. ndm_reset.NDMRESET_CLUSTER_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `ndmreset_cluster_count` | R | 0x0 | Number of external CPU clusters connected on the low bits of NDMRESET_REQUEST and NDMRESET_PROCESS. Software masks those registers with this value. The maximum is 32. |

## SMC Miscellaneous Wrapper Address Map

Aggregates the SMC scratch, chip-configuration, and NDM-reset register blocks.

**NOTE**

**Register Arrays:**

- **scratch_cold.SCRATCH[8]**: 8 registers, base 0x0, stride 0x4
- **scratch_cold_warm.SCRATCH[8]**: 8 registers, base 0x80, stride 0x4

*Table 2450. SMC Miscellaneous Wrapper Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0x1C | scratch_cold.SCRATCH[8] | RW | - |
| 0x80 - 0x9C | scratch_cold_warm.SCRATCH[8] | RW | - |
| 0x100 | VERSION_LO | R | - |
| 0x104 | VERSION_HI | R | - |
| 0x108 | CHIP_ID | R | - |
| 0x10C | LC_STATE | R | - |
| 0x200 | NDMRESET_REQUEST | R | - |
| 0x204 | NDMRESET_PROCESS | RW | - |
| 0x208 | NDMRESET_CLUSTER_COUNT | R | - |

### scratch_cold.SCRATCH[8]

*Table 2451. misc_wrap.scratch_cold.SCRATCH[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x0 | Scratch register |

### scratch_cold_warm.SCRATCH[8]

*Table 2452. misc_wrap.scratch_cold_warm.SCRATCH[8] fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `data` | RW | 0x0 | Scratch register |

### VERSION_LO

*Table 2453. misc_wrap.VERSION_LO fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `version_lo` | R | 0x100A0 | SMC version, lower 32 bits |

### VERSION_HI

*Table 2454. misc_wrap.VERSION_HI fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `version_hi` | R | 0x0 | SMC version, upper 32 bits |

### CHIP_ID

*Table 2455. misc_wrap.CHIP_ID fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `chip_id` | R | 0x0 | Chip ID (e.g. 0 = Memory, 1 = CPU, 2 = AI Compute, 3 = Ethernet) |

### LC_STATE

*Table 2456. misc_wrap.LC_STATE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7:0 | `lc_state` | R | 0xF | Lifecycle state value from SEP. Default to invalid state |

### NDMRESET_REQUEST

*Table 2457. misc_wrap.NDMRESET_REQUEST fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `ndmreset_request` | R | 0x0 | Bit i is the NDM reset request from external CPU cluster i. Bits below ndmreset_cluster_count follow the request input. Bits at and above that count read as 0. The field holds up to 32 clusters. |

### NDMRESET_PROCESS

*Table 2458. misc_wrap.NDMRESET_PROCESS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | `ndmreset_process` | RW | 0x0 | Bit i is the firmware response to the NDM reset request from external CPU cluster i. Bits below ndmreset_cluster_count are driven on the process output. Bits at and above that count are stored and are not driven. The field holds up to 32 clusters. |

### NDMRESET_CLUSTER_COUNT

*Table 2459. misc_wrap.NDMRESET_CLUSTER_COUNT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 7:0 | `ndmreset_cluster_count` | R | 0x0 | Number of external CPU clusters connected on the low bits of NDMRESET_REQUEST and NDMRESET_PROCESS. Software masks those registers with this value. The maximum is 32. |

### Base Config and DFX Status

### SMC Base Configuration Address Map

Base-address, region-size, per-block clock-gating enable, and interrupt configuration registers.

*Table 2460. SMC Base Configuration Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | GLOBAL_BASE | RW | - |
| 0x8 | LOCAL_BASE | R | - |
| 0x10 | REGION_SIZE | RW | - |
| 0x18 | CLOCK_GATE_CONTROL | RW | - |
| 0x20 | HANG_DET_SYS_AXI_CTRL | RW | AXI hang detector control and interrupt status. |
| 0x28 | HANG_DET_SYS_AXI_TIMEOUT_THRESHOLD | RW | Consecutive cycles with outstanding AXI transactions and no completion before a timeout fires. |
| 0x30 | HANG_DET_SEP_AXI_CTRL | RW | AXI hang detector control and interrupt status. |
| 0x38 | HANG_DET_SEP_AXI_TIMEOUT_THRESHOLD | RW | Consecutive cycles with outstanding AXI transactions and no completion before a timeout fires. |
| 0x40 | HANG_DET_DATA_ACCEL_CTRL | RW | AXI hang detector control and interrupt status. |
| 0x48 | HANG_DET_DATA_ACCEL_TIMEOUT_THRESHOLD | RW | Consecutive cycles with outstanding AXI transactions and no completion before a timeout fires. |

#### GLOBAL_BASE

*Table 2461. smc_base_config.GLOBAL_BASE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `base` | RW | 0x40000000 | Global base address for the CPU |

#### LOCAL_BASE

*Table 2462. smc_base_config.LOCAL_BASE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:56 | `Reserved` | - | - | Reserved |
| 55:0 | `base` | R | 0xC0000000 | Local base address for the CPU |

#### REGION_SIZE

*Table 2463. smc_base_config.REGION_SIZE fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:32 | `Reserved` | - | - | Reserved |
| 31:0 | `size` | RW | 0x1000000 | Region size for SMC resources, sizing both the local alias window and the global aperture. Must be a non-zero power of two: the fabric derives its local-alias address mask as (size - 1), so any other value produces a discontiguous mask and misroutes accesses. Zero leaves no reachable local aperture. GLOBAL_BASE and LOCAL_BASE must be aligned to this size. Inbound accesses to a unit whose window extends past this size answer DECERR. Defaults to 16MB (0x100_0000); FW reprograms per chiplet spec. |

### CLOCK_GATE_CONTROL

*Table 2464. smc_base_config.CLOCK_GATE_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:30 | `Reserved` | - | - | Reserved |
| 29:24 | `cg_hysteresis` | RW | 0x1F | Clockgate cg_hysteresis |
| 23:14 | `Reserved` | - | - | Reserved |
| 13 | `telemetry_cg_en` | RW | 0x0 | Clock gate enable for telemetry. While set, the telemetry receiver clocks are stopped and every access to the SMC_TELEMETRY_RECEIVER_WRAP window answers SLVERR with read data 0xBADCAB1E. |
| 12 | `uart_cg_en` | RW | 0x0 | Clock gate enable for UART. While set, the UART peripheral clock is stopped and every access to the SMC_UART_WRAP window answers SLVERR with read data 0xBADCAB1E. |
| 11 | `i2c_cg_en` | RW | 0x0 | Clock gate enable for I2C. While set, the I2C peripheral clock is stopped and every access to the SMC_I2C_WRAP window answers SLVERR with read data 0xBADCAB1E. |
| 10 | `avs_cg_en` | RW | 0x0 | Clock gate enable for AVSBus. While set, the AVSBus controller clocks are stopped and every access to the SMC_AVSBUS_CONTROLLER window answers SLVERR with read data 0xBADCAB1E. |
| 9 | `i3c_cg_en` | RW | 0x0 | Clock gate enable for I3C. While set, the I3C peripheral clock is stopped and every access to the OCA_I3C_WRAP window answers SLVERR with read data 0xBADCAB1E. |
| 8 | `zeroer_cg_en` | RW | 0x0 | Clock gate enable for axi zeroer |
| 7 | `output_fabric_cg_en` | RW | 0x0 | Clock gate enable for the output fabric remap demux and mux |
| 6 | `addr_remap_cg_en` | RW | 0x0 | Clock gate enable for address remapper register logic |
| 5 | `filter_ib_reg_cg_en` | RW | 0x0 | Clockgate enable for inbound filter registers |
| 4 | `filter_ib_axi_cg_en` | RW | 0x0 | Clockgate enable for the inbound filters |
| 3 | `filter_ob_reg_cg_en` | RW | 0x0 | Clockgate enable for outbound filter registers |
| 2 | `filter_ob_axi_cg_en` | RW | 0x0 | Clockgate enable for the outbound filters |
| 1 | `mailbox_cg_en` | RW | 0x0 | Clockgate enable for the mailbox |
| 0 | `dma_cg_en` | RW | 0x0 | Clockgate enable for the DMA |

### HANG_DET_SYS_AXI_CTRL

*Table 2465. smc_base_config.HANG_DET_SYS_AXI_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12 | `irq` | R | 0x0 | Reads 1 while this detector's interrupt is asserted, from a timeout or from irq_test, and 0 while enable or irq_en is clear. A live level, not a latch: it clears when the interrupt drops. Writes are ignored. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `irq_test` | RW | 0x0 | Interrupt test. When 1, asserts irq_o without a bus stall, so firmware can exercise the interrupt path. Gated by enable and irq_en: it substitutes for the timeout, it does not override the interrupt enable. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `irq_en` | RW | 0x0 | Interrupt enable. When irq_en and enable are high, irq_o follows the detector; when 0, irq_o is held low. Gates a real timeout and irq_test alike. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `enable` | RW | 0x0 | When 1, the timeout counter runs. When 0, the counter reloads from the threshold and irq_o is forced low. |

### HANG_DET_SYS_AXI_TIMEOUT_THRESHOLD

*Table 2466. smc_base_config.HANG_DET_SYS_AXI_TIMEOUT_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `value` | RW | 0x1000 | Threshold in clock cycles. Default 0x1000. 20 bits ⇒ up to ~1M cycles (~1ms at 1GHz). 0 disables timeout detection: a stalled bus never asserts irq_o, though irq_test still asserts it. A threshold write takes effect when the stall count next restarts (on a completion, an idle bus, or enable cleared). So it does not change a stall already being timed. |

### HANG_DET_SEP_AXI_CTRL

*Table 2467. smc_base_config.HANG_DET_SEP_AXI_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12 | `irq` | R | 0x0 | Reads 1 while this detector's interrupt is asserted, from a timeout or from irq_test, and 0 while enable or irq_en is clear. A live level, not a latch: it clears when the interrupt drops. Writes are ignored. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `irq_test` | RW | 0x0 | Interrupt test. When 1, asserts irq_o without a bus stall, so firmware can exercise the interrupt path. Gated by enable and irq_en: it substitutes for the timeout, it does not override the interrupt enable. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `irq_en` | RW | 0x0 | Interrupt enable. When irq_en and enable are high, irq_o follows the detector; when 0, irq_o is held low. Gates a real timeout and irq_test alike. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `enable` | RW | 0x0 | When 1, the timeout counter runs. When 0, the counter reloads from the threshold and irq_o is forced low. |

### HANG_DET_SEP_AXI_TIMEOUT_THRESHOLD

*Table 2468. smc_base_config.HANG_DET_SEP_AXI_TIMEOUT_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 19:0 | `value` | RW | 0x1000 | Threshold in clock cycles. Default 0x1000. 20 bits ⇒ up to ~1M cycles (~1ms at 1GHz). 0 disables timeout detection: a stalled bus never asserts irq_o, though irq_test still asserts it. A threshold write takes effect when the stall count next restarts (on a completion, an idle bus, or enable cleared). So it does not change a stall already being timed. |

### HANG_DET_DATA_ACCEL_CTRL

*Table 2469. smc_base_config.HANG_DET_DATA_ACCEL_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:13 | `Reserved` | - | - | Reserved |
| 12 | `irq` | R | 0x0 | Reads 1 while this detector's interrupt is asserted, from a timeout or from irq_test, and 0 while enable or irq_en is clear. A live level, not a latch: it clears when the interrupt drops. Writes are ignored. |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `irq_test` | RW | 0x0 | Interrupt test. When 1, asserts irq_o without a bus stall, so firmware can exercise the interrupt path. Gated by enable and irq_en: it substitutes for the timeout, it does not override the interrupt enable. |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `irq_en` | RW | 0x0 | Interrupt enable. When irq_en and enable are high, irq_o follows the detector; when 0, irq_o is held low. Gates a real timeout and irq_test alike. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `enable` | RW | 0x0 | When 1, the timeout counter runs. When 0, the counter reloads from the threshold and irq_o is forced low. |

### HANG_DET_DATA_ACCEL_TIMEOUT_THRESHOLD

*Table 2470. smc_base_config.HANG_DET_DATA_ACCEL_TIMEOUT_THRESHOLD fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:20 | `Reserved` | - | - | Reserved |
| 19:0 | `value` | RW | 0x1000 | Threshold in clock cycles. Default 0x1000. 20 bits ⇒ up to ~1M cycles (~1ms at 1GHz). 0 disables timeout detection: a stalled bus never asserts irq_o, though irq_test still asserts it. A threshold write takes effect when the stall count next restarts (on a completion, an idle bus, or enable cleared). So it does not change a stall already being timed. |

## DFX Control and Status Address Map

Memory-repair and MBIST status, debug clock and cross-trigger controls, and debug-mux selects.

*Table 2471. DFX Control and Status Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | STATUS_SMU | R | - |
| 0x8 | DEBUG_CTRL | RW | - |
| 0x10 | DEBUG_BUS_MUX | RW | Mode, ID and lane selects for the twelve debug-bus muxes that reduce the SMC debug bus to the 16 CLA input lanes. Each mux takes its mode while Dbmid equals its ID, and its lane selects while Dbmid equals its ID and its mode is 1, so program one mux at a time. |

### STATUS_SMU

*Table 2472. dfx_ctrl_status.STATUS_SMU fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:13 | `Reserved` | - | - | Reserved |
| 12 | `mbist_abort` | R | 0x0 | One bit indicator from DFT logic to say that MBIST has hit timeout and aborted This register is sticky 1 and will hold the value until reset |
| 11:9 | `Reserved` | - | - | Reserved |
| 8 | `mbist_pass` | R | 0x0 | One bit indicator from DFT logic to say when MBIST is successful This register is sticky 1 and will hold the value until reset |
| 7:5 | `Reserved` | - | - | Reserved |
| 4 | `mbist_done` | R | 0x0 | One bit indicator from DFT logic to say when MBIST is completed This register is sticky 1 and will hold the value until reset |
| 3 | `Reserved` | - | - | Reserved |
| 2 | `mem_repair_abort` | R | 0x0 | One bit indicator from DFT logic to say that mem repair has hit timeout and aborted This register is sticky 1 and will hold the value until reset |
| 1 | `mem_repair_success` | R | 0x0 | One bit indicator from DFT logic to say if mem repair was successful This register is sticky 1 and will hold the value until reset |
| 0 | `mem_repair_done` | R | 0x0 | One bit indicator from DFT logic to say when mem repair is completed. This register is sticky 1 and will hold the value until reset |

### DEBUG_CTRL

*Table 2473. dfx_ctrl_status.DEBUG_CTRL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31 | `Reserved` | - | - | Reserved |
| 30:23 | `debug_marker` | RW | 0x0 | Holds the written value for software to read back. It has no effect in the SMC; the debug marker on the SMC debug bus comes from the core logic analyzer (CLA). |
| 22:7 | `xtrig_clk_halt_mask` | RW | 0x0 | CLA cross-trigger and halt-clock enable. Bit 0 (register bit 7) enables every CLA cross-trigger output lane together with the CLA halt-clock request; 0 holds them all inactive. Bits 15:1 hold the written value for software to read back and have no effect. |
| 6 | `dtb_ew_en` | RW | 0x0 | Holds the written value for software to read back. It has no effect in the SMC. |
| 5 | `dtb_ns_en` | RW | 0x0 | Holds the written value for software to read back. It has no effect in the SMC. |
| 4 | `force_clk_en` | RW | 0x0 | Design-for-debug (DFD) clock force. 1 keeps the DFD clock running whatever the value of cg_en. |
| 3 | `cg_en` | RW | 0x0 | DFD clock stop. 1 stops the clock to the debug-bus muxes and the CLA unless force_clk_en is 1; 0 lets it run. The clock also runs while the SMC is in reset. |
| 2 | `gpio_en` | RW | 0x0 | Holds the written value for software to read back. It has no effect in the SMC. |
| 1:0 | `chiplet_enable` | RW | 0x0 | Holds the written value for software to read back. It has no effect in the SMC. |

### DEBUG_BUS_MUX

*Table 2474. dfx_ctrl_status.DEBUG_BUS_MUX fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 63:58 | `Muxselseg7` | RW | 0x0 | No effect in the SMC: each debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 57:52 | `Muxselseg6` | RW | 0x0 | No effect in the SMC: each debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 51:46 | `Muxselseg5` | RW | 0x0 | No effect in the SMC: each debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 45:40 | `Muxselseg4` | RW | 0x0 | No effect in the SMC: each debug-bus mux has four output lanes, selected by Muxselseg0 to Muxselseg3. |
| 39:34 | `Muxselseg3` | RW | 0x0 | Binary input-lane select for output lane 3 of the addressed debug-bus mux, which takes eight 16-bit input lanes and drives four. 0 selects input lane 3; a value k from 1 to 4 selects input lane k+3, so 4 selects lane 7; 5 to 63 drive zero. |
| 33:28 | `Muxselseg2` | RW | 0x0 | Binary input-lane select for output lane 2 of the addressed debug-bus mux, which takes eight 16-bit input lanes and drives four. 0 selects input lane 2; a value k from 1 to 4 selects input lane k+3, so 4 selects lane 7; 5 to 63 drive zero. |
| 27:22 | `Muxselseg1` | RW | 0x0 | Binary input-lane select for output lane 1 of the addressed debug-bus mux, which takes eight 16-bit input lanes and drives four. 0 selects input lane 1; a value k from 1 to 4 selects input lane k+3, so 4 selects lane 7; 5 to 63 drive zero. |
| 21:16 | `Muxselseg0` | RW | 0x0 | Binary input-lane select for output lane 0 of the addressed debug-bus mux, which takes eight 16-bit input lanes and drives four. 0 selects input lane 0; a value k from 1 to 4 selects input lane k+3, so 4 selects lane 7; 5 to 63 drive zero. |
| 15:8 | `Rsvd158` | RW | 0x0 | - |
| 7:2 | `Dbmid` | RW | 0x0 | ID of the debug-bus mux to program: 3 to 6 address the level-2 muxes and 7 to 14 the level-3 muxes. |
| 1:0 | `Dbmmode` | RW | 0x0 | Mode selection, 0: DBM off, 1: Normal debug mode, 2: DBM ID output mode, 3: Toggle Mode |

### Component Register References

For detailed register-level documentation of each SMC component, refer to the individual component documentation (Table 2475).

*Table 2475. SMC Component Documentation*

| Category | Description | Reference |
|---|---|---|
| CPU and System Control | CPU Control, PLIC, CLINT, Watchdog Timers, and Bus Error Units | cpu |
| Interrupt Controller | Platform-Level Interrupt Controller (PLIC) details | interrupts |
| DMA Controller | DMA Controller detailed register interface | dma |
| Memory Zeroer | Memory Zeroer control registers | zeroer |
| Fabric Control | Address remapping, inbound/outbound filters, and fabric control registers | fabric |
| Peripherals and I/O | Mailbox, System Timer OCTS, eFuse, UART 16550, Log Engine, I2C, I3C, AVSBus, GPIO, and Telemetry Receiver | periphs |
| Reset Management | Reset unit and system control registers | clk_rst |
| Clock Generation | PLL wrapper and clock generation registers | pll |
| DFD / CLA | Design-for-debug wrap and Core Logic Analyzer register map | dfd |

## 2.1.7. Debug & Test Ports (DTP)

The Debug & Test Ports (DTP) module provides debug and test infrastructure. It integrates the JTAG Interface Unit (JIU) and the Cross Trigger Network (CTN) into a single instantiable subsystem that exposes a standard external JTAG port, internal debug bridges, and a configurable cross-trigger fabric.

### 2.1.7.1. Capabilities

- **JTAG access** — IEEE 1149.1-2013 Primary TAP with configurable Secondary TAPs for hierarchical multi-chiplet debug connectivity per IEEE 1838-2019 (3DIC).
- **iJTAG chains** — IEEE 1687-2014 SIB-based chains for DFT secure, DFT non-secure, and DFD instrumentation, plus SMC and SEP OTP debug bridges.
- **JTAG2AXI bridges** — SMC fabric, SMC OTP and SEP OTP debug access without system software.
- **Cross-trigger fabric** — Configurable network (1–32 external CTPs, 0–32 internal cross triggers) supporting both wire-OR and point-to-point signalling per the Open Chiplet Cross Triggering (OCCT) specification.
- **Clock stop** — Unified halt path combining JTAG DEBUG_CONTROL and CLA clock-stop requests, with separate CLA-only readback for software observation.
- **Lifecycle feature control** — SEP-driven enable polarity gates eleven debug and test paths (five STAP/scan paths, three iJTAG chains and three JTAG2AXI bridges) independently.

### 2.1.7.2. Overview

The DTP contains two major functional blocks — the JTAG Interface Unit (JIU) and the Cross Trigger Network (CTN) — plus the DTP top-level wiring that connects them to each other, to the SMC and SEP, and to the external JTAG and cross-trigger pads.

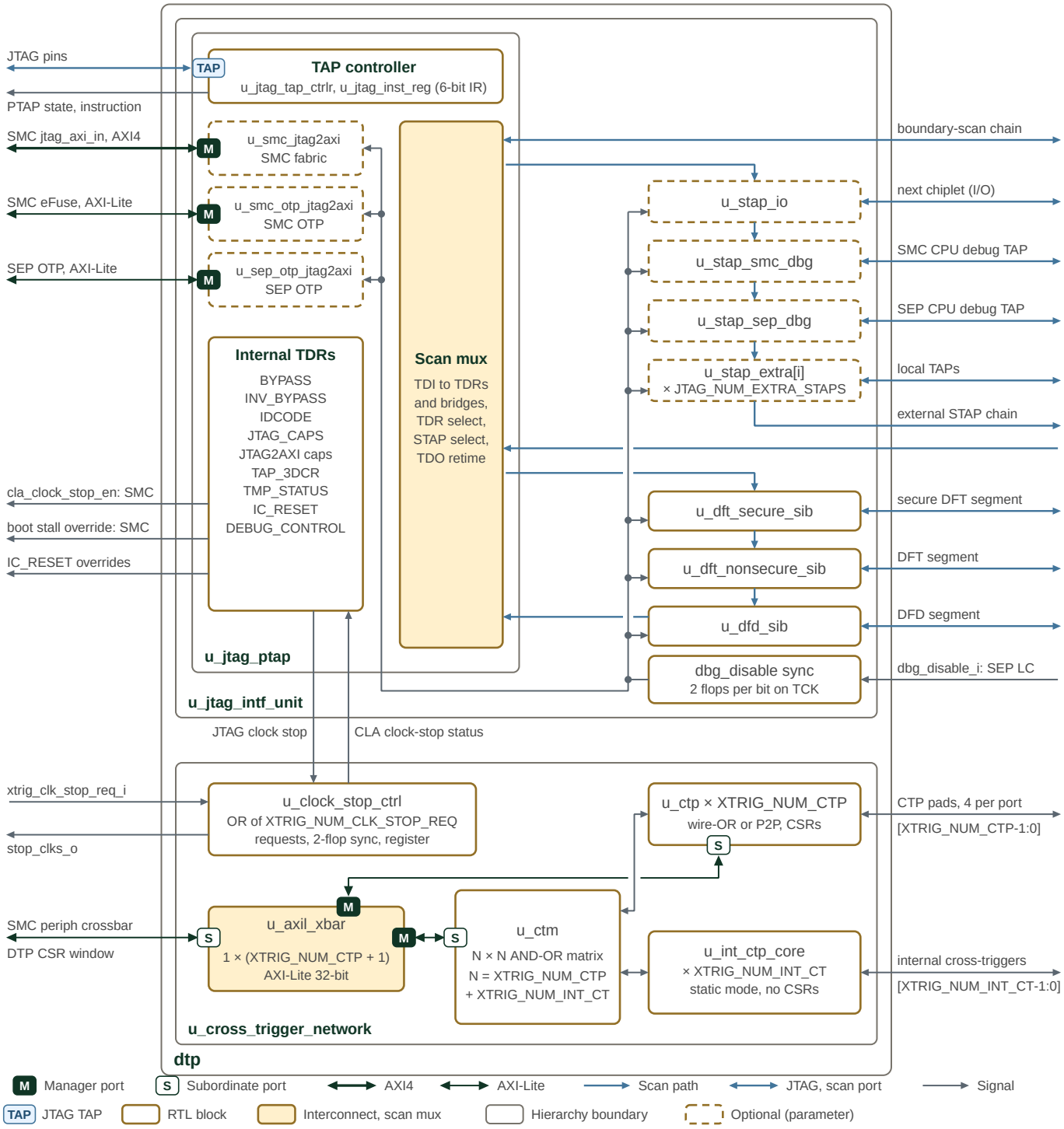


*Figure 78. DTP internal structure*

Figure 78 draws each RTL instance under its instance name inside the `u_jtag_intf_unit` and `u_cross_trigger_network` hierarchy boundaries. Labels at the figure edges name what each DTP port connects to in the SMU. Blue lines carry JTAG and scan data, dark lines AXI4 and AXI-Lite, and grey lines other signals. Dashed blocks exist only when their enable parameter is set: the three JTAG2AXI bridges follow `JTAG_SMC_DBG_ENABLE` and `JTAG_SEP_DBG_ENABLE`, and the STAPs follow `JTAG_STAP_IO_ENABLE`, `JTAG_SMC_DBG_ENABLE`, `JTAG_SEP_DBG_ENABLE` and `JTAG_NUM_EXTRA_STAPS`. The grey fan-out from the debug-disable synchroniser reaches every STAP, SIB and bridge. The DTP JTAG scan chain figure gives the serial order of the scan paths.

The figure names each count by its `dtp` parameter. `XTRIG_NUM_CTP` external CTPs, `XTRIG_NUM_INT_CT` internal lanes and

XTRIG_NUM_CLK_STOP_REQ clock-stop requests default to the dtp_pkg values 16, 10 and 9. The CTM is N × N, where N is XTRIG_NUM_CTP + XTRIG_NUM_INT_CT (26 by default); its generated register geometry must equal that sum, which elaboration checks. u_axil_xbar has one manager port for the CTM and one for each external CTP, 17 by default. JTAG_NUM_EXTRA_STAPS sets the number of u_stap_extra instances, 1 by default. In the cross-trigger network, u_axil_xbar decodes the DTP CSR window, which the SMC peripheral crossbar rebases to 0, onto the CTM at offset 0x000 and one 16-byte window for each external CTP from 0x200. The internal cross-trigger lanes are u_int_ctp_core instances without CSRs, whose protocol XTRIG_INT_CT_MODE fixes. In the SMU the first two lanes connect to the SMC DFD CLA, one clock-stop request comes from the CLA, and the remaining lanes and requests leave the SMU.

#### JTAG Interface Unit

The JTAG Interface Unit (JIU) is the top-level JTAG control module. It contains:

- A Primary TAP (PTAP) — the main chiplet JTAG entry point, connected to the five external JTAG pins (TCK, TDI, TDO, TMS, TRST_n).
- A configurable set of Secondary TAPs (STAPs) — for I/O chiplet-to-chiplet JTAG, SMC CPU debug, SEP CPU debug, and optional extra TAPs.
- An iJTAG network selected by SELECT_IJTAG, with a serial cascade of secure DFT, non-secure DFT, and DFD SIBs. Each SIB inserts or bypasses its associated external scan segment.
- Independent instruction-selected PTAP TDRs for SMC fabric debug (JTAG2AXI), SMC OTP (JTAG2AXIL), and SEP OTP (JTAG2AXIL). These bridges are outside the iJTAG SIB cascade.

#### Cross Trigger Network

The Cross Trigger Network (CTN) routes cross-trigger pulses between on-chip sources and sinks and the external CTP pads. It consists of:

- **Cross Trigger Ports (CTPs)** — pad-facing modules, each connecting to four I/O pads (CT_Req_out, CT_Ack_in, CT_Req_in, CT_Ack_out) and supporting both wire-OR and point-to-point signalling.
- **Cross Trigger Matrix (CTM)** — an N × N AND-OR crossbar (26 × 26 by default) that statically routes trigger pulses under AXI-Lite CSR control.
- **Clock stop aggregation** — the CTN OR-reduces all xtrig_clk_stop_req_i inputs (CLA sources) and OR-combines the result with the JTAG jtag_clock_stop bit to produce the registered stop_clks_o output.

Per-path debug-disable inputs gate STAPs, iJTAG SIBs, and JTAG2AXI bridges.

#### Debug Access Paths

External debug access reaches the DTP subsystem through two routes:

**JTAG path**: An external debugger drives the five JTAG pins directly. The PTAP decodes the JTAG instruction, selects the appropriate TDR, and routes data through the scan chain. JTAG2AXI instructions allow the debugger to generate AXI transactions on the SMC fabric via the PTAP's built-in bridges, without involving the SMC CPU.

**AXI path**: The DTP cross-trigger configuration registers (CTM and CTP CSRs) are memory-mapped inside the SMC address space. The SMC exposes them on its axil_dtp_csr_req_o / axil_dtp_csr_resp_i interface, which the SMU wires directly to the DTP axil_xtrig_req_i / axil_xtrig_resp_o ports. Software running on the SMC reaches the CTN CSRs through its own register map (dtp_ctrl_reg in smc.rdl). An external debugger using the PTAP JTAG2AXI bridge reaches the same window through the SMC fabric AXI port. See the CTN Memory Map section for the CTN-internal address layout.

Firmware and an external debugger share this window. The Programmer's Guide gives the cross-trigger programming sequence.

### 2.1.7.3. Clocks and Resets

#### Clock Domains

The DTP operates in two clock domains (Table 2476).

*Table 2476. DTP clock domains*

| Domain | Signal | Description |
|---|---|---|
| System clock | `clk_i` | Main clock for cross-trigger logic and AXI interfaces. All CTN and CSR logic runs in this domain. |
| JTAG clock | TCK | JTAG test clock for all TAP controller and scan logic. Clocks the PTAP, STAP state machines, and scan registers. Asynchronous to `clk_i`. |

The combined JTAG and CLA clock-stop request crosses into `clk_i` through two synchronizer stages in `ctn_clock_stop_ctrl`, followed by an output register. The CLA clock-stop status also crosses into TCK through the DEBUG_CONTROL TDR's two-flop `u_cla_clock_stop_sync` synchronizer before bit 4 captures it. The CTP handshake crossings are described in the CTP interface.

#### Reset Architecture

Table 2477 identifies the reset inputs and the logic they reset.

*Table 2477. DTP reset signals*

| Signal | Description |
|---|---|
| `rst_n_i` | Active-low asynchronous reset for cross-trigger logic and AXI interfaces. Resets all CTN and CSR state; clock-stop output resets to 0 (clocks running). |
| `pwr_on_rst_ni` | Active-low asynchronous power-on reset for JTAG logic. Combined internally with TRST_n; resets the PTAP to Test-Logic-Reset state and clears all TDRs. |
| TRST_n (JTAG pin) | Optional external active-low TAP reset. When asserted, drives the PTAP to Test-Logic-Reset regardless of the TCK or TMS state. |

All registers reset to safe states. The JTAG TAP returns to Test-Logic-Reset on either TRST_n or power-on reset.

#### IC_RESET Overrides

The primary TAP exposes a reset-selection test data register (TDR) through `IC_RESET`. IEEE 1149.1-2013 defines the instruction in Clause 8.21 and the reset-selection register in Clause 17. The IEEE working-group summary of the 2013 changes identifies both clauses. In this implementation, per reset port it holds `reset_control`, the active-low reset value to drive, and `reset_enable`, which selects it. One `reset_hold` bit sits nearest TDO.

`reset_enable` is active low despite the name: it resets to 1, leaving the port's functional reset connected, and writing 0 hands the port over to `reset_control`. The overrides leave the DTP on `jtag_ic_reset_{smc,sep,ext}_o`, whose `.ovrd` member is active high and whose `.val` carries `reset_control`.

`reset_hold` also resets to 1, which lets Test-Logic-Reset clear the enable/control bits. Write 0 to keep an override asserted across TLR; only TRST clears `reset_hold` itself.

Consumers select between the two sources with a hazard-free multiplexer, which holds its output while the select alone

moves. Moving `reset_enable` and `reset_control` in one Update-DR changes the select and a data input together, which can pulse a reset on a port that neither source is resetting. The Programmer's Guide gives the override sequence.

At the SMU the TDR concatenates three slices from TDI to TDO: SMC, SEP, then the external slice, followed by `reset_hold`. The SMC and SEP port lists are OCAH-defined. The external slice is adopter-defined: its width and fields come from the SMU type parameter `ic_reset_ext_t`. The Integrator Guide records the SMU-composed port map.

### Clock Stop

The DTP combines JTAG DEBUG_CONTROL and CLA clock-stop request sources into one functional chiplet halt path, while keeping the CLA-only status separate for debugger readback.

### Halt Path Architecture

Figure 79 shows how JTAG and CLA requests contribute to the chiplet halt path, with the DTP as the SMU instantiates it. The DEBUG_CONTROL TDR is drawn as its five bits, with bit 0 nearest TDO. The grey box below the DTP is the part of the SMC DFD block (`smc_dfd_wrap`) that produces the SMC's clock-stop request and exchanges cross-triggers with the DTP. Gate and flop symbols draw the logic of the path, and CDC badges mark the two-flop synchronizers where a signal enters the TCK or `clk_i` domain.

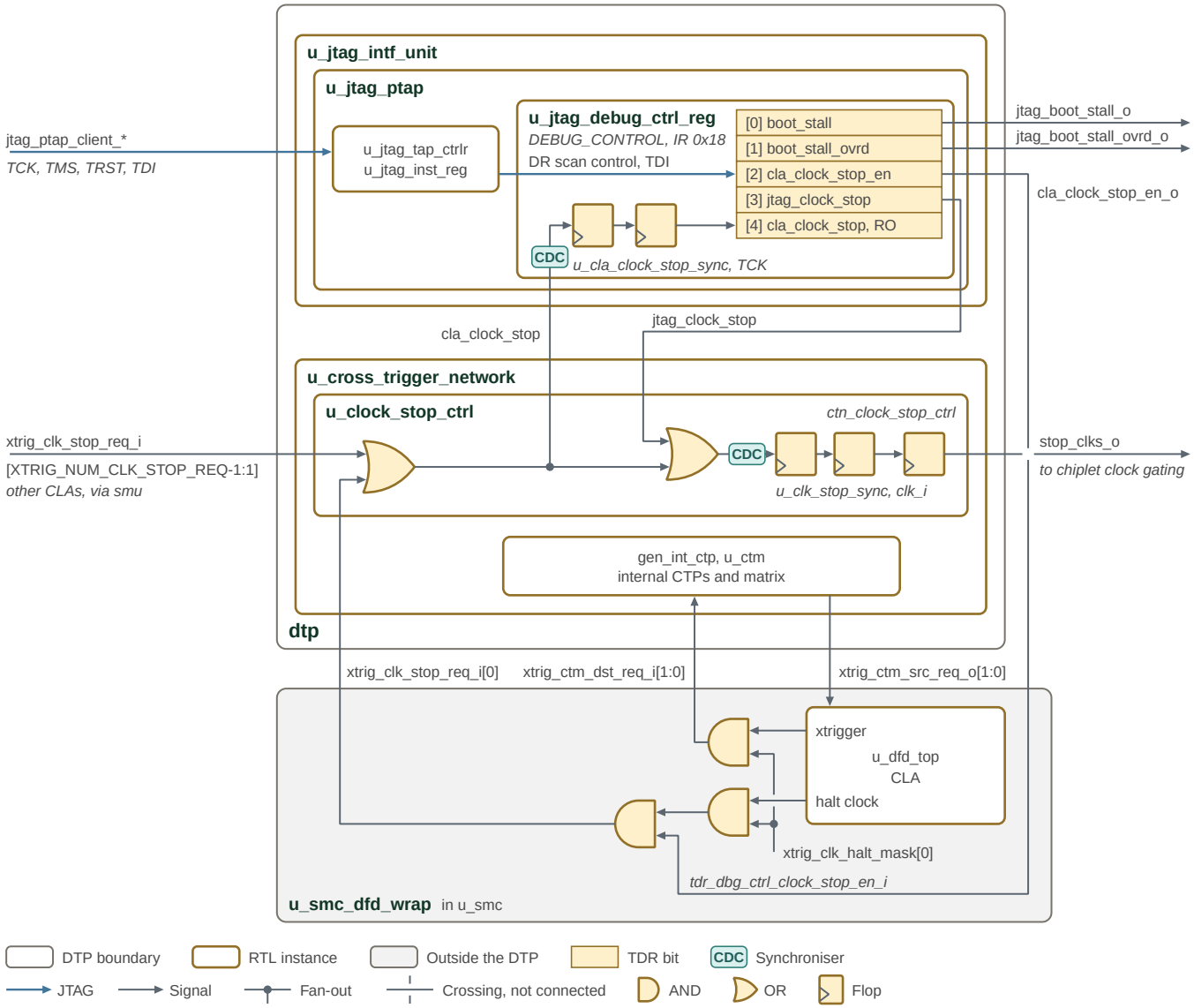


*Figure 79. DTP JTAG and CLA clock-stop aggregation*

Three distinct signals participate in clock stop:

**`jtag_clock_stop` (JTAG DEBUG_CONTROL bit 3)**

Written through the DEBUG_CONTROL TDR (IR 0x18) in the primary TAP and routed to the CTN clock-stop controller. OR-combined with CLA requests to form the registered `stop_clks_o` output that connects to chiplet functional clock gating. The bit resets on TRST, on power-on reset and whenever the TAP enters Test-Logic-Reset.

**`cla_clock_stop_en` (JTAG DEBUG_CONTROL bit 2)**

Exported at the DTP top level as `cla_clock_stop_en_o`. It is not part of the `stop_clks_o` aggregation inside the CTN. In the SMU it drives the SMC DFD's `tdr_dbg_ctrl_clock_stop_en_i`, which gates the SMC CLA's clock-stop request on `xtrig_clk_stop_req_i[0]`, so it decides whether the SMC CLA can halt the chiplet.

**CLA clock-stop indication**

The CTN OR-reduces `xtrig_clk_stop_req_i` (CLA sources only, not `jtag_clock_stop`) to form the `cla_clock_stop` readback status. The OR is combinational, so the DEBUG_CONTROL TDR passes it through a two-flop TCK synchronizer before bit 4 captures it in Capture-DR. A debugger reads it there to observe whether any CLA clock-stop source is active.

#### Signal Flow

1. The JTAG Interface Unit captures DEBUG_CONTROL bits in the TCK domain. The `jtag_clock_stop` bit is routed to the CTN clock-stop controller; `cla_clock_stop_en` is exported at the DTP boundary.
2. The CTN OR-reduces `xtrig_clk_stop_req_i` to produce `cla_clock_stop` status, then OR-combines that aggregate with `jtag_clock_stop`. The result passes a two-flop `clk_i` synchronizer and an output flop to produce `stop_clks_o`, which follows the requests three `clk_i` cycles later and is low while `rst_n_i` is asserted.
3. The `cla_clock_stop` status is fed back into the DEBUG_CONTROL readback path so a debugger can observe CLA clock-stop source activity independently of its own direct-stop assertion.

This architecture keeps CLA observability distinct from the JTAG direct-stop path, while either source can assert the functional chiplet halt through `stop_clks_o`.

In the SMU, lane 0 of the `XTRIG_NUM_CLK_STOP_REQ` `xtrig_clk_stop_req_i` lanes carries the SMC DFD's clocks-stopped output. That output is the CLA halt-clock action, masked by bit 0 of `xtrig_clk_halt_mask` in the SMC DFX CSRs and ANDed with `cla_clock_stop_en_o`; it also raises an SMC CPU interrupt. The remaining lanes, `xtrig_clk_stop_req_i[XTRIG_NUM_CLK_STOP_REQ-1:1]`, come from the SMU's own `xtrig_clk_stop_req_i` ports. The same mask gates the CLA cross-trigger outputs, which reach the cross-trigger matrix through internal lanes 1 and 0 (`xtrig_ctm_dst_req_i[1:0]`); `xtrig_ctm_src_req_o[1:0]` returns to the CLA. The SMU runs both SMC lanes in pulse-synchronization mode and ties their acknowledges low, and exports internal lanes 9 to 2.

For the CTN clock-stop controller implementation and the `xtrig_clk_stop_req_i` input indexing, see Cross-Trigger Network.

#### Clock Distribution

Figure 80 shows which DTP blocks use the system clock and which use TCK. The DTP has two clocks and no clock gates. `clk_i`, the system clock (`clk_smu_i` in the SMU), clocks the whole cross-trigger network and the AXI side of the three JTAG2AXI bridges. TCK, from `jtag_ptap_client_tap_ctrl_i.tck`, clocks the rest of the JTAG interface unit. Each block is tinted by the clock that runs it, and each clock has one arrow into each hierarchy it reaches. CDC badges mark the blocks that synchronize a signal from another domain: the cross-trigger port input synchronizers, the clock-stop controller, the `dbg_disable_i` synchronizers, the DEBUG_CONTROL `cla_clock_stop` synchronizer and the JTAG2AXI bridges' `u_axi_cdc`. TCK leaves the DTP unchanged on the host TAP control of every STAP and on the host scan control of the boundary-scan, extended STAP, DFT and DFD ports.

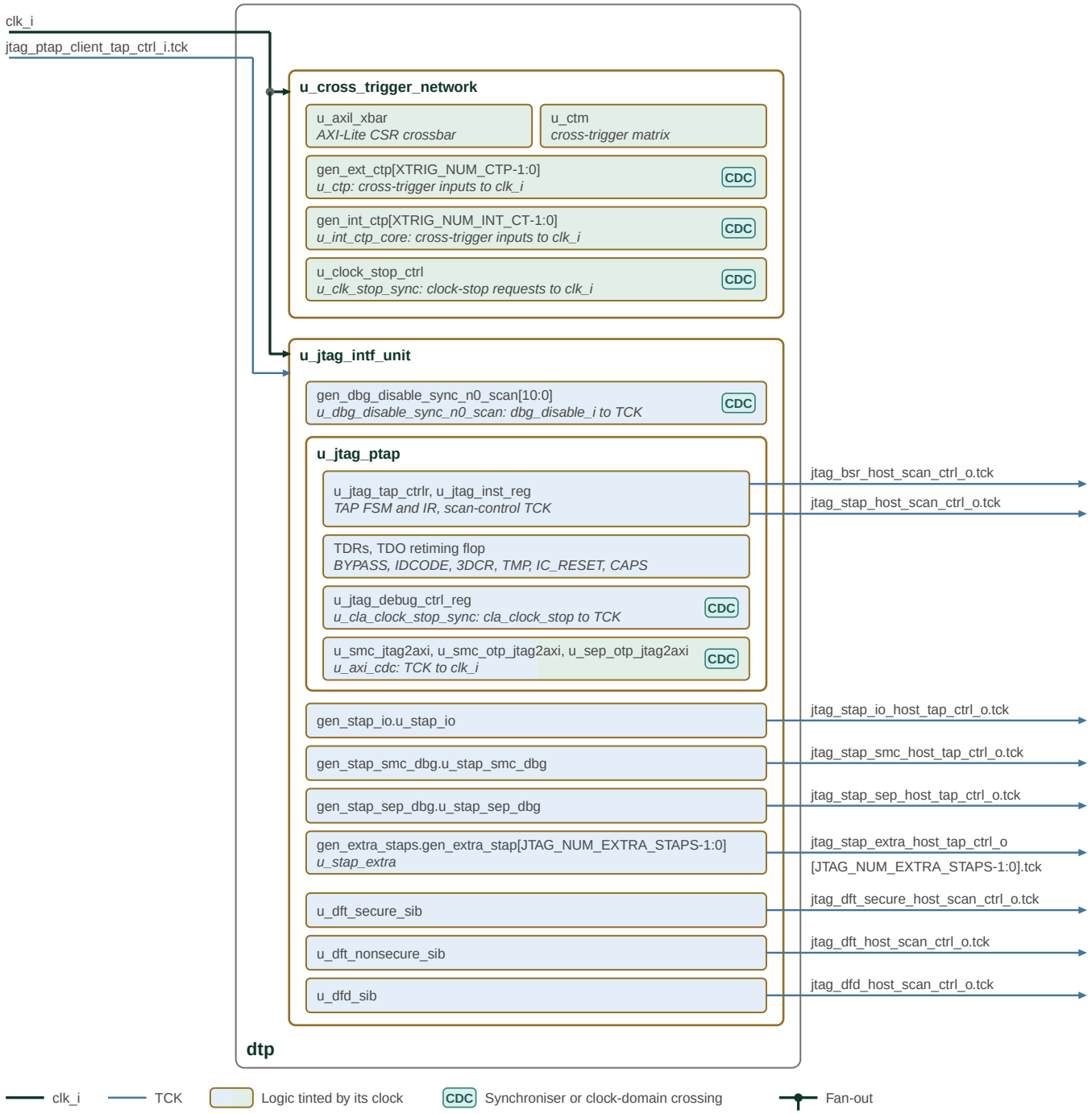


*Figure 80. DTP system-clock and JTAG-clock distribution*

### Reset Distribution

Figure 81 shows the reset distribution for DTP system logic and the JTAG scan paths. Each reset has its own colour, and the input names below the DTP ports are the SMU nets that drive them. Gate and flop symbols show where resets are combined, and a reset ends on the bottom edge of the flop it resets.

- `rst_n_i` (`rst_primary_smc_clk_no` in the SMU) resets the cross-trigger network and the AXI side of the JTAG2AXI bridges.
- `pwr_on_rst_ni` (`powergood_stable` in the SMU) sets the `dbg_disable_i` synchronizers to 1, disabling every debug path until the lifecycle value arrives. `u_trst_n_and` ANDs it with the TRST pin to form the combined TRST.

- The combined TRST resets the TAP controller, including its Test-Logic-Reset flop, the TMP controller and the TDO retiming flop. It also resets the hold flops of the 3DCR and IC_RESET TDRs, the 3DCR scan register and the STAPs, and leaves unchanged on the host TAP control of every STAP.
- The TAP controller derives the scan reset `rst_n`, which is low in Test-Logic-Reset, and the scan channel reset `chrst_n`, which the TMP persistence mode holds released through Test-Logic-Reset. `rst_n` resets the instruction register, most TDRs, the TCK side of the JTAG2AXI bridges, the STAPs and the SIBs. The 3DCR TDR takes it only while its hold bit is clear, and the IC_RESET TDR only while its `reset_hold` bit is set. `chrst_n` is not used inside the DTP; both leave on the host scan control of the boundary-scan, extended STAP, DFT and DFD ports.

The two-flop `cla_clock_stop` synchronizer in the DEBUG_CONTROL TDR has no reset. `test_en_i` drives the test mode of the cross-trigger network and the JTAG2AXI bridges. `scan_rst_ni` is not connected inside the DTP, which has no reset synchronizer, so `rst_n_i` must arrive scan-controlled. A STAP whose enable parameter is 0 drives its host TRST low.

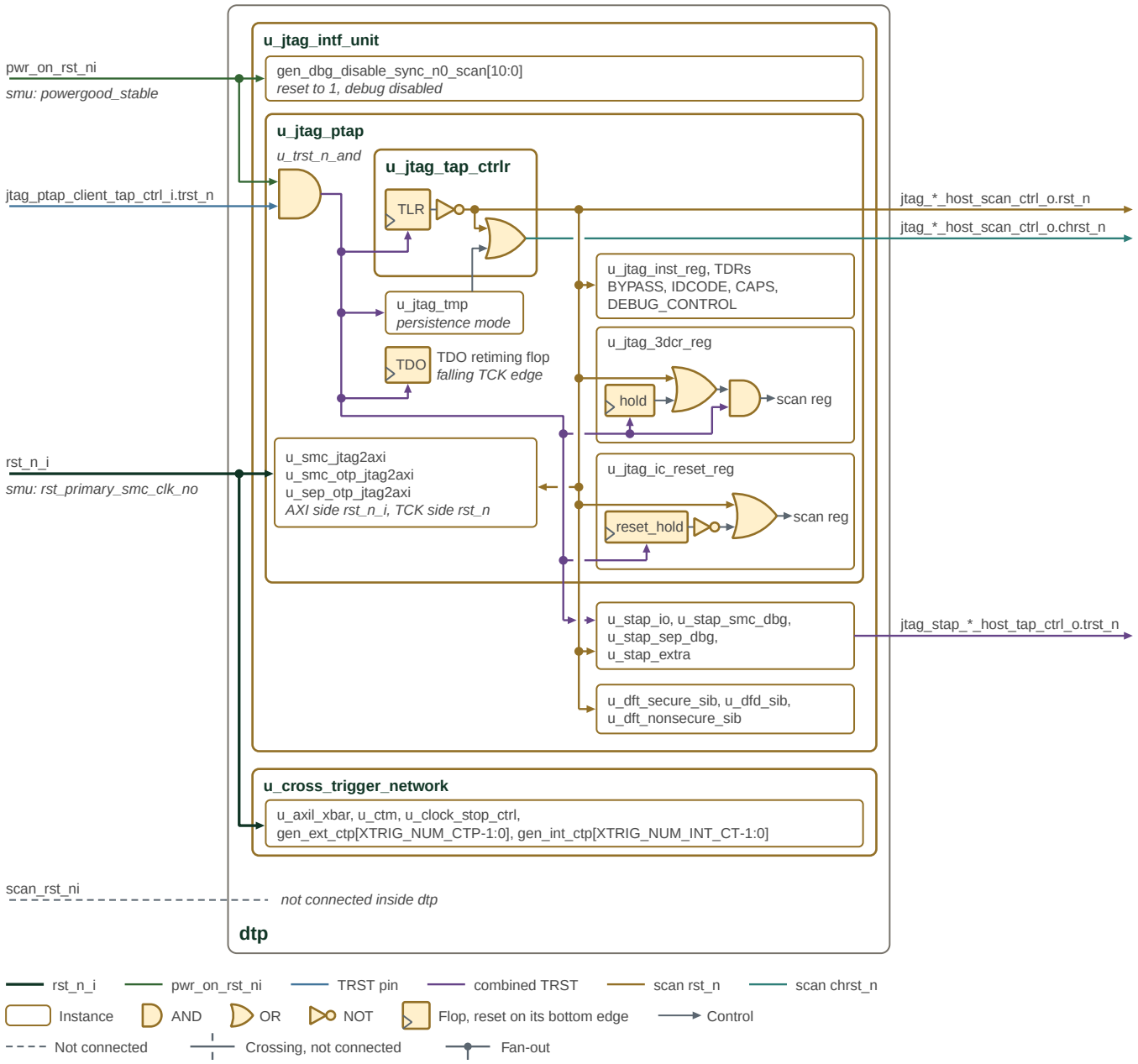


*Figure 81. DTP system and JTAG reset distribution*

### 2.1.7.4. JTAG Operation

This page describes how the DTP integrates its JTAG components at the subsystem level: topology, scan chain ordering, lifecycle feature control. Clock domains and reset overrides are described in Clocks and Resets; IP Reference contains the individual JTAG components.

#### DTP JTAG Topology

The DTP instantiates a single JTAG Interface Unit (JIU) as the primary debug entry point for the chiplet. The JIU connects to:

- The five external JTAG pins (TCK, TDI, TDO, TMS, TRST_n).
- The Cross Trigger Network, for the `jtag_clock_stop` bit in the DEBUG_CONTROL instruction and for the `cla_clock_stop` readback status.
- The SMC AXI fabric via a JTAG2AXI bridge.
- The SMC and SEP OTP controllers via AXI4-Lite JTAG2AXI bridges.
- The SMC and SEP CPU debug TAPs via STAPs.

Figure 82 shows the PTAP, secondary scan path, and iJTAG instrument network.

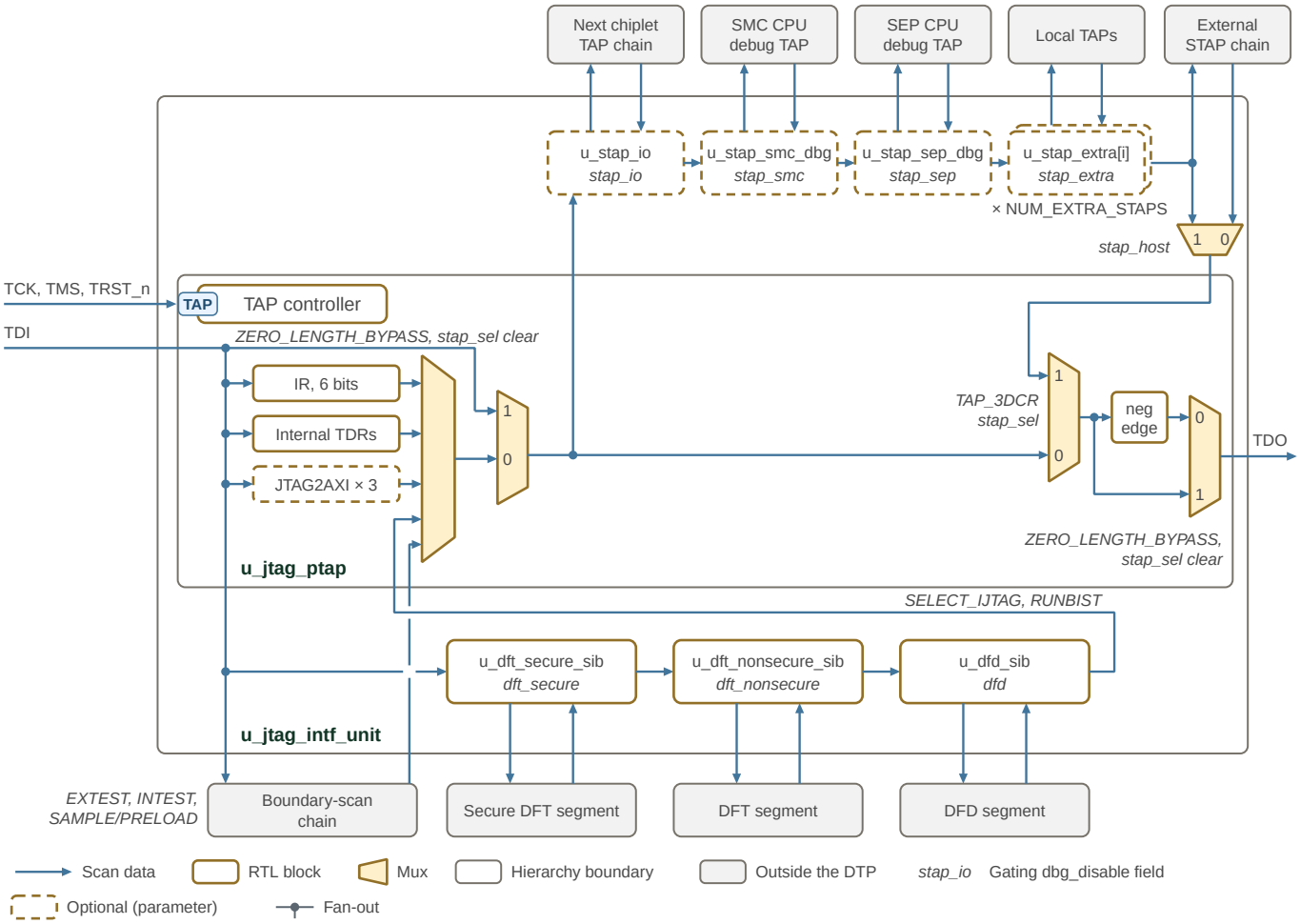


*Figure 82. DTP JTAG scan chain*

Read Figure 82 along the arrows from TDI to TDO. TDI reaches every register and external chain at once; the IR and TDR

mux returns the instruction register in Shift-IR and, in Shift-DR, the register the current instruction selects. Italic labels beside a mux name the instructions or the select bit that choose its input, and 0 or 1 at a mux input is the select value that picks it. The PTAP output then enters the STAP chain, which is scanned only while the `TAP_3DCR` STAP-select bit is set: the bit gates the chain's scan strobes and selects its return onto TDO. Otherwise TDO takes the PTAP output directly. TDO is retimed on the falling edge of TCK except during a `ZERO_LENGTH_BYPASS` DR scan with the STAP-select bit clear, when the last mux takes the TDO mux output ahead of the retimer and TDI reaches TDO combinationally. While the bit is set, `ZERO_LENGTH_BYPASS` behaves exactly as `BYPASS`: the PTAP bypass register leads the STAP chain and the chain's return is retimed. The figure counts the extra STAPs by the `jtag_intf_unit` parameter `NUM_EXTRA_STAPS`, which `dtp` sets from `JTAG_NUM_EXTRA_STAPS`. Each STAP and SIB carries, in italics, the `dbg_disable_i` field that closes it. Dashed blocks exist only when their enable parameter is set, as the list below gives for the STAPs; the JTAG2AXI bridges follow `JTAG_SMC_DBG_ENABLE` and `JTAG_SEP_DBG_ENABLE`. Each STAP adds one SIB bit to the chain; opening the SIB inserts the STAP's 3-bit 3DCR, and setting the 3DCR STAP-select bit splices in its secondary TAP. Each iJTAG SIB inserts its external segment after its SIB bit while that bit is set.

#### STAP Secondary Scan Path

The PTAP secondary scan path connects the STAPs in this fixed order:

1. PTAP
2. I/O STAP — when `JTAG_STAP_IO_ENABLE=1`
3. SMC debug STAP — when `JTAG_SMC_DBG_ENABLE=1`
4. SEP debug STAP — when `JTAG_SEP_DBG_ENABLE=1`
5. Extra STAP instances `[JTAG_NUM_EXTRA_STAPS-1:0]`
6. Return to PTAP

When a STAP is disabled by parameter, the DTP bypasses it in the secondary scan chain and drives its host outputs low, apart from TCK, which still follows the PTAP.

#### JTAG2AXI Bridges

The PTAP contains three JTAG2AXI bridges, each selected by its own group of six instructions (Table 2478). The request and response type parameters of a bridge set its bus type, address width and data width. Its read and write pipeline-depth parameters set how many transactions it queues, where 0 allows a single outstanding transaction. The table lists the DTP defaults. Each bridge reports its configuration in its `*_JTAG2AXI_CAPS` register. Every output of a bridge's manager port depends only on the bridge's internal state, so no combinational path runs from the port's inputs to its outputs. A bridge accepts a response from the cycle after its request handshake completes.

*Table 2478. DTP JTAG2AXI bridges*

| Bridge | Instructions | Bus | Address / data width (bits) | Type parameters | Pipeline-depth parameters (default 3) |
|---|---|---|---|---|---|
| SMC fabric | 0x27–0x2C (`SMC_*`) | AXI4 | 56 / 64 | `smc_jtag_axi_req_t`, `smc_jtag_axi_resp_t` | `SMC_RD_PL_DEPTH`, `SMC_WR_PL_DEPTH` |
| SMC OTP | 0x1B–0x20 (`SMC_OTP_*`) | AXI4-Lite | 32 / 32 | `smc_otp_axil_req_t`, `smc_otp_axil_resp_t` | `SMC_OTP_RD_PL_DEPTH`, `SMC_OTP_WR_PL_DEPTH` |
| SEP OTP | 0x21–0x26 (`SEP_OTP_*`) | AXI4-Lite | 32 / 32 | `sep_otp_axil_req_t`, `sep_otp_axil_resp_t` | `SEP_OTP_RD_PL_DEPTH`, `SEP_OTP_WR_PL_DEPTH` |

#### Debug Disable

`dbg_disable_i` (`sep_lifecycle_ctrl_pkg::dbg_disable_t`) is a per-path debug/test disable vector. Each field is active-high (1

= path disabled). The JTAG interface unit synchronizes the vector into the TCK domain and applies each field to the named path.

Table 2479 maps each debug path to its active-high disable field.

*Table 2479. DTP debug-disable paths*

| Path | Disable Field |
|---|---|
| I/O STAP host path | `dbg_disable_i.stap_io` |
| SMC debug STAP host path | `dbg_disable_i.stap_smc` |
| SEP debug STAP host path | `dbg_disable_i.stap_sep` |
| Extra STAP host paths | `dbg_disable_i.stap_extra` |
| External STAP return path | `dbg_disable_i.stap_host` |
| Secure external DFT iJTAG path | `dbg_disable_i.dft_secure` |
| Non-secure external DFT iJTAG path | `dbg_disable_i.dft_nonsecure` |
| DFD iJTAG path | `dbg_disable_i.dfd` |
| SMC fabric JTAG2AXI path | `dbg_disable_i.smc_jtag2axi` |
| SMC OTP JTAG2AXI path | `dbg_disable_i.smc_otp_jtag2axi` |
| SEP OTP JTAG2AXI path | `dbg_disable_i.sep_otp_jtag2axi` |

A disabled path remains observable only where required for control and readback. For a JTAG2AXI bridge, disabling means TDR update registers are not latched. The disable gates the idle state: an active AXI transaction continues to completion, and buffered requests are flushed when idle is reached. For a STAP, the host-visible scan access is clamped so the disabled path cannot reach downstream secure logic.

The STAP and iJTAG fields are applied in `jtag_intf_unit`; the three JTAG2AXI fields are applied in `jtag_ptap`.

### 2.1.7.5. Cross-Trigger Operation

The Cross-Trigger Network (CTN) routes debug events between internal sources and external cross-trigger ports. Matrices select event destinations; ports provide wire-OR or point-to-point signaling at the chiplet boundary.

#### Event Routing

The CTN combines Cross-Trigger Matrices (CTMs) and Cross-Trigger Ports (CTPs). The port modes are configured before the matrix routes for an event path, and both before the participating debug sources are enabled.

#### Clock Stop

JTAG and CLA clock-stop requests feed the chiplet halt path. Clocks and Resets owns its aggregation, clock-domain crossing and reset behavior.

Clock-stop integration

#### Programming Reference

IP Reference groups the network, port and matrix descriptions. The CTN owns the unified register map; its component chapters describe routing and handshakes.

CTN Register Map

The Programmer's Guide gives the cross-trigger programming sequence for the port modes and matrix routes referenced above.

### 2.1.7.6. IP Reference

The JTAG family provides scan and debug access. The cross-trigger family routes events and coordinates debug actions between blocks and chiplets.

#### JTAG

#### JTAG Interface Unit

The JTAG Interface Unit (JIU) is the top-level JTAG control module for the DTP subsystem. It manages the primary JTAG port and all supporting instrument logic.

#### Overview

The JIU contains:

- A Primary TAP (PTAP) with chiplet IDCODE, boundary scan, BIST, reset control and JTAG2AXI bridges.
- A configurable number of Secondary TAPs (STAPs) for downstream chiplet JTAG, SMC CPU debug, SEP CPU debug, and optional additional TAPs.
- An iJTAG network with SIB-based chains for DFT (secure and non-secure), DFD, and JTAG2AXI bridges to the SMC fabric, SMC OTP and SEP OTP controllers.
- Per-path debug-disable inputs gate STAPs, iJTAG SIBs, and JTAG2AXI bridges.

The PTAP, STAP and all supporting logic conform to IEEE 1149.1-2013 (JTAG), IEEE 1687-2014 (iJTAG) and IEEE 1838-2019 (3DIC).

#### Configuration Parameters

The JIU is configured through the PTAP feature enables, the I/O, SMC debug and SEP debug STAP enables, the number of additional STAPs, the IDCODE and JTAG_CAPS values, and the JTAG2AXI pipeline depths.

The parameters of `jtag_intf_unit` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Architecture

The JTAG Interface Unit integrates the PTAP, STAP chain and iJTAG network into a unified debug access point.

Figure 83 shows the PTAP, STAP chain, and iJTAG network at the debug access point. Follow the arrows from TDI to TDO: italic labels beside a mux name the instructions or select bit that choose its input, 0 or 1 at a mux input is the select value that picks it, and italic labels in the STAPs and SIBs name the `dbg_disable_i` field that closes each one. Dashed blocks exist only when their enable parameter is set; `NUM_EXTRA_STAPS` sets the number of `u_stap_extra` instances. TDO is retimed on the falling edge of TCK except during a `ZERO_LENGTH_BYPASS` DR scan with the `TAP_3DCR` STAP-select bit clear, when the last mux takes the TDO mux output ahead of the retimer and TDI reaches TDO combinationally. While the bit is set, `ZERO_LENGTH_BYPASS` behaves exactly as `BYPASS`: the PTAP bypass register leads the STAP chain and the chain's return is retimed, because the chain launches its last bit on the rising TCK edge.

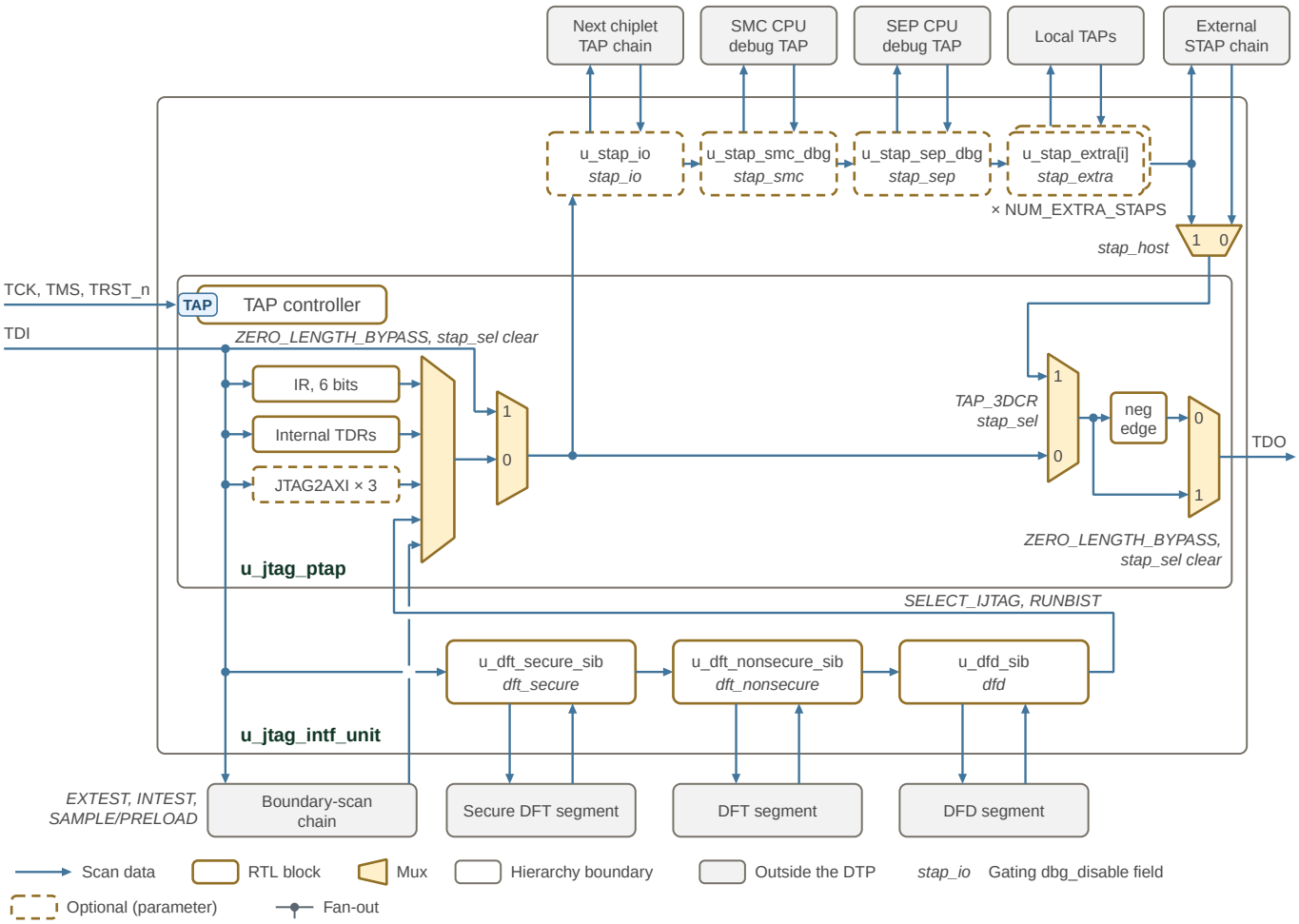


*Figure 83. JTAG Interface Unit scan chain*

### Scan Chain Topology

The PTAP is connected to the incoming JTAG interface for the chiplet. The scan chain is organized in three major segments: the Primary TAP processes all instructions, the Secondary TAP chain provides hierarchical TAP access, and the iJTAG network enables instrument-level access to debug blocks.

### STAP Chain Hierarchy

The STAP chain implements a four-level hierarchy with configurable lockup latch insertion optimized for different physical routing distances.

Table 2480 relates each STAP level to its lockup-latch configuration and physical role.

*Table 2480. JIU STAP chain hierarchy*

| Level | STAP Type | TDI_LOCKUP | SCAN_OUT_LOCKUP | Purpose |
|---|---|---|---|---|
| 1 | I/O STAP | 1 | 0 | Chiplet-to-chiplet connectivity (die boundary crossing) |
| 2 | SMC Debug STAP | 0 | 0 | On-chip connection to SMC CPU debugger TAP |
| 3 | SEP Debug STAP | 0 | 0 | On-chip connection to SEP CPU debugger TAP |
| 4 | Extra STAPs [N-1:0] | 0 | 0 | Local debug ports |

The I/O STAP enables TDI lockup to handle the longer propagation delays across die boundaries. On-chip STAPs (SMC Debug, SEP Debug) disable TDI lockup since signal routing is local to the chiplet.

No STAP in the chain enables scan output lockup. The chain return reaches the PTAP's falling-edge TDO retimer, and a lockup ahead of that retimer would add one TCK of latency to every scan through the STAP chain. The last STAP's scan output also drives `stap_host_scan_out_o`, which launches on the rising edge of TCK like the host scan outputs of the iJTAG SIBs.

`stap_host_scan_out_o` carries the scan bit of the last STAP's SIB. That bit resets to 0 on TRST and in Test-Logic-Reset, so the port reads 0 from reset.

#### Debug Disable Synchronization

`dbg_disable_i` (`sep_lifecycle_ctrl_pkg::dbg_disable_t`) is in the system clock domain and must be synchronized to TCK before use. The interface unit synchronizes each bit through a 2-FF chain whose reset value is 1 (fail-closed). The synchronized fields gate STAP and SIB selection and are forwarded to the PTAP for the JTAG2AXI bridges.

Each field is active-high (1 = path disabled).

Table 2481 maps each debug access point to its synchronized disable field.

*Table 2481. JIU debug-disable fields*

| Access Point | Disable Field |
|---|---|
| I/O STAP | `dbg_disable_i.stap_io` |
| SMC Debug STAP | `dbg_disable_i.stap_smc` |
| SEP Debug STAP | `dbg_disable_i.stap_sep` |
| Extra STAPs | `dbg_disable_i.stap_extra` |
| External STAP return path | `dbg_disable_i.stap_host` |
| DFT Secure SIB | `dbg_disable_i.dft_secure` |
| DFT Non-Secure SIB | `dbg_disable_i.dft_nonsecure` |
| DFD SIB | `dbg_disable_i.dfd` |
| SMC fabric JTAG2AXI | `dbg_disable_i.smc_jtag2axi` |
| SMC OTP JTAG2AXI | `dbg_disable_i.smc_otp_jtag2axi` |
| SEP OTP JTAG2AXI | `dbg_disable_i.sep_otp_jtag2axi` |

#### iJTAG Network

The iJTAG network is organized into three cascaded Segment Insertion Bit (SIB) chains. Each SIB is gated by the corresponding synchronized debug-disable field. Each SIB's scan bit and control bit reset to 0 on TRST and in Test-Logic-Reset, so `dft_secure_host_scan_out_o`, `dft_host_scan_out_o` and `dfd_host_scan_out_o` read 0 from reset, whether or not the SIB is disabled.

The PTAP selects the iJTAG network as its DR under SELECT_IJTAG and, when `RUNBIST_ENABLE` is set, under RUNBIST. From TDI the scan path passes through the DFT Secure SIB, the DFT Non-Secure SIB and the DFD SIB, and the DFD SIB's output returns to the PTAP. Each SIB is one bit long. While a SIB is open, its segment is inserted on the TDO side of its bit: the SIB bit drives the segment's scan input, and the segment's return continues to the next SIB. With every SIB closed the iJTAG DR is three bits long, laid out as in Table 2482.

*Table 2482. JIU iJTAG DR with every SIB closed*

| DR Bit | SIB | Segment Ports | Disable Field |
|---|---|---|---|
| 0 | DFD | `dfd_host_scan_*` | `dbg_disable_i.dfd` |
| 1 | DFT Non-Secure | `dft_host_scan_*` | `dbg_disable_i.dft_nonsecure` |
| 2 | DFT Secure | `dft_secure_host_scan_*` | `dbg_disable_i.dft_secure` |

Bit 0 is nearest TDO and is the first bit shifted in. A SIB bit shifted in as 1 opens that SIB at Update-DR, and its segment is on the scan path from the next DR scan. Each SIB captures its own open state. A SIB whose disable field is set stays closed and remains a one-bit stage that captures 0, whatever is shifted into it.

With the DFT Non-Secure SIB open and an N-bit segment, the DR is N + 3 bits long: bit 0 is the DFD SIB, bits N to 1 are the segment with the bit nearest its scan output at bit 1, bit N + 1 is the DFT Non-Secure SIB and bit N + 2 is the DFT Secure SIB. The network attached to each segment's ports defines its contents and length. The Programmer's Guide gives the segment access sequence.

#### DFT Secure Chain

The DFT Secure chain is gated by `dbg_disable_i.dft_secure`. This port is for security-sensitive DFT control.

#### DFT Non-Secure Chain

The DFT Non-Secure chain is gated by `dbg_disable_i.dft_nonsecure`. General DFT control is attached to this port.

#### DFD Chain

The Design-for-Debug (DFD) chain is gated by `dbg_disable_i.dfd`. Integrator-defined debug instruments are attached to this port.

#### JTAG2AXI Interfaces

The interface unit instantiates multiple JTAG2AXI bridges. Each bridge is a set of PTAP TDRs selected by its own instructions and gated by its own `dbg_disable_i` field (Table 2481); no bridge is reached through the iJTAG network.

- **SMC JTAG2AXI TDRs**: This set of TDRs provide a mechanism to inject AXI transactions onto the SMC AXI local interconnect.
- **SMC Fuse JTAG2AXIL TDRs**: This set of TDRs provide a mechanism to interact directly with the SMC fuse controller. The JTAG2AXIL interface connects to the fuse controller CSRs through an AXI4-lite-to-APB bridge.
- **SEP Fuse JTAG2AXIL TDRs**: This set of TDRs provide a mechanism to interact directly with the SEP fuse controller. The JTAG2AXIL interface connects to the fuse controller CSRs through an AXI4-lite-to-APB bridge.

### Interface

#### Instruction Encodings

The JTAG interface unit uses 6-bit instruction encodings (Table 2483).

*Table 2483. JIU instruction encodings*

| Encoding | Name | Register Size | Security |
|---|---|---|---|
| 0x00 | BYPASS (IEEE 1149.1, Section 8.4) | 1 | |
| 0x01 | IDCODE (IEEE 1149.1, Section 8.13) | 32 | |

| Encoding | Name | Register Size | Security |
|---|---|---|---|
| 0x02 | RUNBIST (IEEE 1149.1, Section 8.10) | BIST | |
| 0x03 | SAMPLE_PRELOAD (IEEE 1149.1, Sections 8.6 and 8.7) | BSR | |
| 0x04 | EXTEST (IEEE 1149.1, Section 8.8) | BSR | |
| 0x05 | EXTEST_TRAIN | BSR | |
| 0x06 | EXTEST_PULSE | BSR | |
| 0x07 | CLAMP (IEEE 1149.1, Section 8.11) | 1 | |
| 0x08 | HIGHZ (IEEE 1149.1, Section 8.16) | 1 | |
| 0x09 | INTEST (IEEE 1149.1, Section 8.9) | BSR | |
| 0x0A | CLAMP_HOLD (IEEE 1149.1, Section 8.20) | 2 | |
| 0x0B | CLAMP_RELEASE (IEEE 1149.1, Section 8.20) | 2 | |
| 0x0C | TMP_STATUS (IEEE 1149.1, Section 8.20) | 2 | |
| 0x0D | IC_RESET (IEEE 1149.1, Section 8.21) | Variable | |
| 0x0E | TAP_3DCR (IEEE 1838, Section 5.5.1) | 2 | |
| 0x10-0x17 | *Reserved for RISC-V* | | |
| 0x18 | DEBUG_CONTROL | 5 | |
| 0x19 | JTAG_CAPS | 60 | |
| 0x1A | SELECT_IJTAG | Variable | |
| 0x1B | SMC_OTP_JTAG2AXI_CAPS | 14 | |
| 0x1C | SMC_OTP_AXI_SINGLE_OP | Variable | |
| 0x1D | SMC_OTP_AXI_SERIES_CTRL | Variable | |
| 0x1E | SMC_OTP_AXI_SERIES_DATA_INCR | Variable | |
| 0x1F | SMC_OTP_AXI_SERIES_DATA_NO_INCR | Variable | |
| 0x20 | SMC_OTP_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | |
| 0x21 | SEP_OTP_JTAG2AXI_CAPS | 14 | |
| 0x22 | SEP_OTP_AXI_SINGLE_OP | Variable | |
| 0x23 | SEP_OTP_AXI_SERIES_CTRL | Variable | |
| 0x24 | SEP_OTP_AXI_SERIES_DATA_INCR | Variable | |
| 0x25 | SEP_OTP_AXI_SERIES_DATA_NO_INCR | Variable | |
| 0x26 | SEP_OTP_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | |
| 0x27 | SMC_JTAG2AXI_CAPS | 14 | |
| 0x28 | SMC_AXI_SINGLE_OP | Variable | |
| 0x29 | SMC_AXI_SERIES_CTRL | Variable | |
| 0x2A | SMC_AXI_SERIES_DATA_INCR | Variable | |
| 0x2B | SMC_AXI_SERIES_DATA_NO_INCR | Variable | |
| 0x2C | SMC_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | |

| Encoding | Name | Register Size | Security |
|---|---|---|---|
| 0x3D | ZERO_LENGTH_BYPASS | 0; 1, as BYPASS, while the PTAP TAP_3DCR STAP-select bit is set | |
| 0x3E | INV_BYPASS | | |
| 0x3F | BYPASS (IEEE 1149.1, Section 8.4) | 1 | |

The length of each variable-length JTAG2AXI register follows from the bridge's address width, data width and pipeline depth, and for the series data registers from the size last written to `*_AXI_SERIES_CTRL`. The PTAP JTAG2AXI Support section defines each layout.

**NOTE:** All instruction encodings *not* explicitly defined in this table map to the BYPASS instruction.

### Signal Interface

Packed control and bus types remain bundles at the `jtag_intf_unit` boundary; their members are not separate scalar ports.

The parameters and ports of `jtag_intf_unit` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Dependencies

The JTAG Interface Unit depends on the following components:

- `hw/ip/jtag/jtag_ptap` - Primary TAP
- `hw/ip/jtag/jtag_stap` - Secondary TAP
- `hw/common/ocah_prim` - Primitive cells

## JTAG PTAP

The Primary TAP (PTAP) is the main JTAG access point for the chiplet. It implements IEEE 1149.1-2013 with extensions for IEEE 1687-2014 (iJTAG) and IEEE 1838-2019 (3DIC).

### Overview

The PTAP provides:

- Full IEEE 1149.1-2013 TAP controller with 16-state FSM and one-hot encoding.
- Test Data Registers (TDRs) for all defined JTAG instructions.
- Test Mode Persistence (TMP) support — gates JTAG clock stop and test assertion across a configurable list of reset conditions.
- Boundary scan with scan ports and control signals.
- BIST support via the RUNBIST instruction.
- 3DIC support (TAP_3DCR) for STAP selection over a JTAG scan chain.
- iJTAG SIB-based network access.
- JTAG2AXI debug bridges for SMC fabric, SMC OTP and SEP OTP.

### Configuration Parameters

The PTAP is configured through feature enables for the optional instructions, TMP, IC_RESET slices and debug bridges, the IC_RESET slice types, the IDCODE and JTAG_CAPS values, and the JTAG2AXI pipeline depths.

The parameters of `jtag_ptap` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Architecture

The PTAP is fully compliant with the IEEE JTAG specifications. It implements a modular design with a core TAP controller, dedicated register modules for each instruction type, and optional AXI bridge modules for debug access. The architecture supports Test Mode Persistence (TMP), boundary scan with dedicated scan ports and control signals, BIST functionality, and 3DIC support with STAP scan ports and control signals derived from internal TDRs.

Figure 84 shows the control, decode, and scan-data paths through the PTAP.

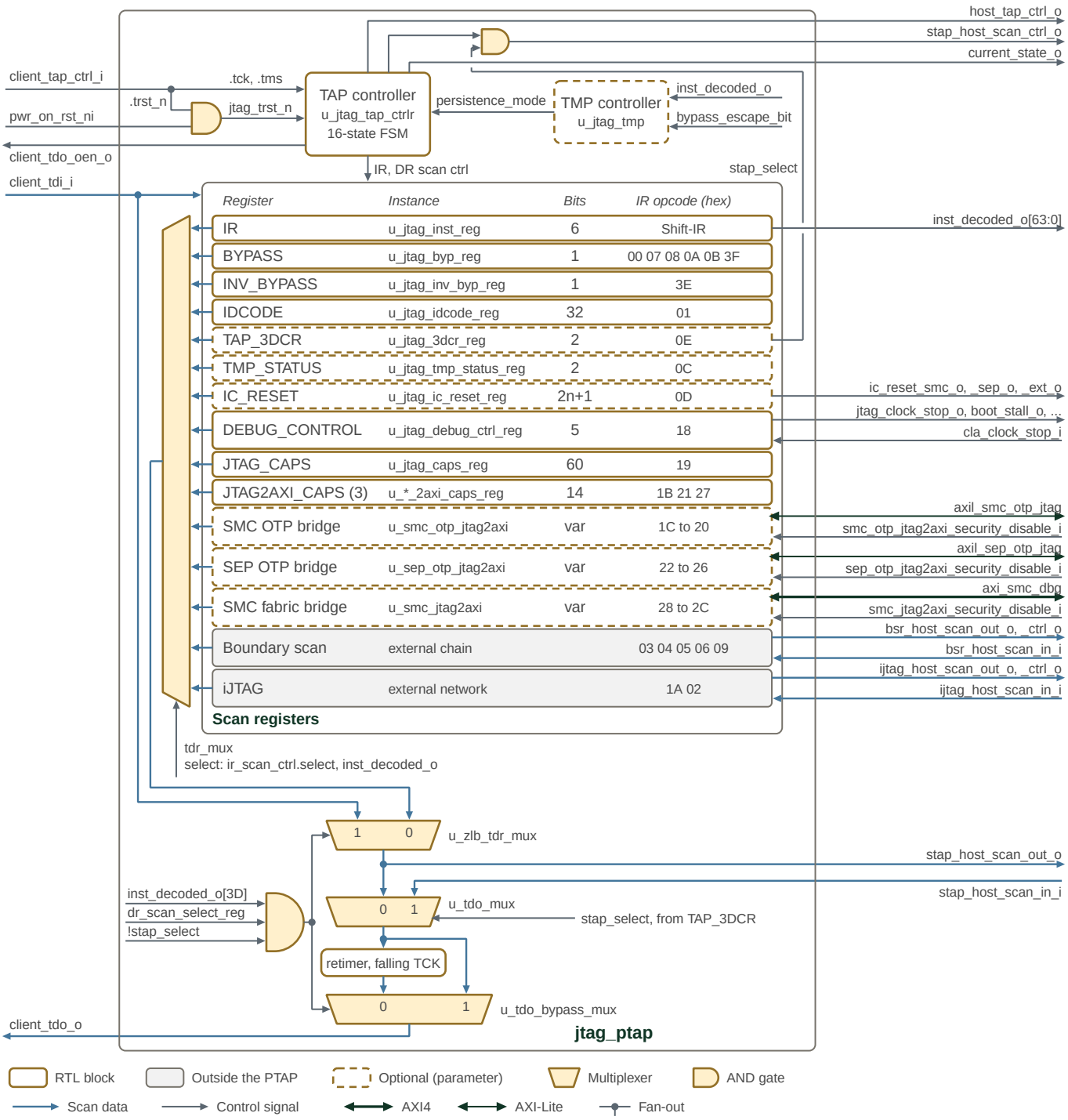


*Figure 84. PTAP module architecture*

Blue lines carry scan data and grey lines carry control. `client_tdi_i` feeds the instruction register and every register in the scan-register group, whose last column gives the IR opcodes, in hex, that select each register for a DR scan. The 6-bit instruction register captures `6'b000001`, resets to IDCODE and decodes its opcode one-hot onto `inst_decoded_o[63:0]`. An opcode with no register of its own, or one whose enable parameter is 0, selects BYPASS. Dashed registers and bridges are present only when their parameter enables them (Table 2484). The boundary-scan chain and the iJTAG network lie outside the PTAP: `client_tdi_i` is forwarded to them, and their returns enter the TDR multiplexer.

`tdr_mux` takes the output of every register in the group and the boundary-scan and iJTAG returns. It selects the instruction register while `ir_scan_ctrl.select` is set, and otherwise the register that `inst_decoded_o` selects. The numbers on the inputs of each 2:1 multiplexer are the select values that pass them. `u_zlb_tdr_mux` passes `client_tdi_i` instead of

tdr_mux during a ZERO_LENGTH_BYPASS DR scan, and its output also leaves on stap_host_scan_out_o toward the STAP chain. u_tdo_mux passes the chain return stap_host_scan_in_i while stap_select, the TAP_3DCR select bit, is set. The retimer registers the result on the falling TCK edge, and u_tdo_bypass_mux bypasses the retimer during a ZERO_LENGTH_BYPASS DR scan. Both ZERO_LENGTH_BYPASS multiplexers take their select from inst_decoded_o[3D] ANDed with dr_scan_select_reg, the DR scan select registered on the falling TCK edge, and with the inverse of stap_select. While stap_select is set, ZERO_LENGTH_BYPASS therefore behaves exactly as BYPASS (Zero-Length Bypass).

pwr_on_rst_ni is ANDed with the client TRST in u_trst_n_and. The result, jtag_trst_n, resets the TAP controller and the retimer, and the controller forwards it with TCK and TMS on host_tap_ctrl_o. The controller also drives the IR and DR scan controls, ORed together and gated by stap_select, on stap_host_scan_ctrl_o, so the STAP chain holds its state while the bit is clear. The TMP controller takes the decoded CLAMP_HOLD, CLAMP_RELEASE and BYPASS instructions and the TMP_STATUS bypass_escape_bit, and returns persistence_mode to the TAP controller. The AXI side of each JTAG2AXI bridge runs on clk_i and rst_n_i and takes the DFT test-mode enable test_en_i; the figure omits all three. Each bridge has its own *_security_disable_i input (Table 2499).

### Module Hierarchy

Table 2484 identifies the PTAP submodules and the conditions under which they are instantiated.

*Table 2484. PTAP module hierarchy*

| Module | Purpose | Conditional |
|---|---|---|
| jtag_tap_ctrlr | IEEE 1149.1 TAP controller FSM (16 states, one-hot encoded) | Always instantiated |
| jtag_inst_reg | 6-bit instruction register with 64-wide one-hot decoded output | Always instantiated |
| jtag_byp_reg | 1-bit bypass register (BYPASS instruction) | Always instantiated |
| jtag_inv_byp_reg | 1-bit inverted bypass register (INV_BYPASS instruction) | Always instantiated |
| jtag_idcode_reg | 32-bit IDCODE register | Always instantiated |
| jtag_3dcr_reg | 2-bit 3D Control Register for STAP control | When STAP_IO_ENABLE \|\| SEP_DBG_ENABLE \|\| SMC_DBG_ENABLE \|\| NUM_EXTRA_STAPS != 0 |
| jtag_tmp | Test Mode Persistence controller | When TMP_ENABLE=1 |
| jtag_tmp_status_reg | 2-bit TMP status register | When TMP_ENABLE=1 |
| jtag_ic_reset_reg | IC_RESET control register (2*(NumSmcIcReset+NumSepIcReset+NumExtIcReset)+1 bits) | When any of IC_RESET_SMC_ENABLE, IC_RESET_SEP_ENABLE, IC_RESET_EXT_ENABLE is 1 |
| jtag_debug_ctrl_reg | 5-bit debug control register | Always instantiated |
| jtag_caps_reg | 60-bit capabilities readback register | Always instantiated |
| jtag2axi (SMC OTP) | JTAG-to-AXI-Lite bridge for SMC OTP controller | When SMC_DBG_ENABLE=1 |
| jtag2axi (SEP OTP) | JTAG-to-AXI-Lite bridge for SEP OTP controller | When SEP_DBG_ENABLE=1 |
| jtag2axi (SMC fabric) | JTAG-to-AXI bridge for SMC system fabric | When SMC_DBG_ENABLE=1 |

### TAP Controller State Machine

The TAP controller implements the IEEE 1149.1 16-state finite state machine using one-hot encoding. Each state is assigned a unique bit position within a 16-bit vector, enabling direct state detection via single-bit comparisons.

Table 2485 lists the one-hot encodings and functions of the sixteen TAP-controller states.

*Table 2485. PTAP state encodings*

| State | Encoding | Description |
|---|---|---|
| TEST_LOGIC_RESET | 16'h0001 | State 0: Reset state, instruction register loads IDCODE |
| RUN_TEST_IDLE | 16'h0002 | State 1: Idle state between operations |
| SELECT_DR_SCAN | 16'h0004 | State 2: Select data register scan path |
| CAPTURE_DR | 16'h0008 | State 3: Capture data into selected DR |
| SHIFT_DR | 16'h0010 | State 4: Shift data through selected DR |
| EXIT1_DR | 16'h0020 | State 5: Exit from shift to either pause or update |
| PAUSE_DR | 16'h0040 | State 6: Pause DR shift operation |
| EXIT2_DR | 16'h0080 | State 7: Exit from pause back to shift or update |
| UPDATE_DR | 16'h0100 | State 8: Update DR from shift register |
| SELECT_IR_SCAN | 16'h0200 | State 9: Select instruction register scan path |
| CAPTURE_IR | 16'h0400 | State 10: Capture data into IR (fixed pattern 6'b000001) |
| SHIFT_IR | 16'h0800 | State 11: Shift data through IR |
| EXIT1_IR | 16'h1000 | State 12: Exit from shift to either pause or update |
| PAUSE_IR | 16'h2000 | State 13: Pause IR shift operation |
| EXIT2_IR | 16'h4000 | State 14: Exit from pause back to shift or update |
| UPDATE_IR | 16'h8000 | State 15: Update IR, decode new instruction |

Five consecutive TCK cycles with TMS high bring the state machine to TEST_LOGIC_RESET from any state, as the IEEE 1149.1 state graph defines. A state register value that is not one of the sixteen one-hot encodings also moves to TEST_LOGIC_RESET on the next rising edge of TCK.

Control signals are generated using a split approach. The update and run-test-idle signals are decoded combinationally from the current state as Moore machine outputs. The capture, shift, test-logic-reset and TDO output enable signals are decoded from the current state and registered on the falling edge of TCK, so they are stable before the next rising edge samples data.

### Reset Hierarchy

The PTAP employs a three-level reset hierarchy to support power-on initialization, JTAG reset, and conditional register preservation.

Table 2486 identifies the source and scope of each PTAP reset.

*Table 2486. PTAP reset hierarchy*

| Reset Signal | Source | Scope |
|---|---|---|
| `jtag_trst_n` | `pwr_on_rst_ni && client_tap_ctrl_i.trst_n` | Primary reset for all JTAG logic, combines power-on and external TRST |
| `test_logic_reset` | Derived from TAP FSM state (asserted when state == TEST_LOGIC_RESET) | Resets instruction and data registers |
| `chrst_n` | `TMP_ENABLE ? (!test_logic_reset \|\| persistence_mode_i) : !test_logic_reset` | Channel reset for TMP-aware registers, preserves state when in persistence mode |

The primary reset `jtag_trst_n` is formed by AND-gating the power-on reset input with the external TRST signal. This combined reset feeds all JTAG clock domain logic. The TAP controller derives `test_logic_reset` from the FSM state, which normally resets all registers. When Test Mode Persistence is enabled, the conditional reset `chrst_n` allows boundary scan and configuration registers to survive TEST_LOGIC_RESET state transitions while in persistence mode.

### TDR Multiplexer

The PTAP implements an instruction-dependent data register multiplexer. During IR scan operations, the instruction register connects to the scan path. During DR scan operations, the appropriate data register is selected based on the currently decoded instruction. Each test data register has dedicated scan control logic that enables selection only when the corresponding instruction is active and the required feature is enabled.

The scan path supports zero-length bypass through explicit `prim_stdmux2` multiplexer cells rather than behavioral multiplexing. This implementation prevents glitches on scan path selectors during instruction transitions.

STAP selection is implemented as a second-level multiplexer. When the 3DCR register's select bit is asserted, the STAP chain input replaces the TDR multiplexer output. The STAP scan control signals are generated by OR-combining the instruction register and data register scan controls, enabling STAP operation during both IR and DR scan sequences as required by IEEE 1838. The select bit also gates the select, capture, shift and update strobes of the STAP scan interface, so they stay low while the bit is clear.

### Zero-Length Bypass

While the TAP_3DCR STAP-select bit is clear, the ZERO_LENGTH_BYPASS instruction connects TDI directly to TDO without intermediate register stages, providing minimum scan latency for hierarchical JTAG chains. The DR scan adds no bit, and TDO follows TDI combinationally rather than through the retimer.

While the STAP-select bit is set, ZERO_LENGTH_BYPASS behaves exactly as BYPASS. The last stage of the STAP chain shifts on the rising TCK edge, so its return must reach TDO through the falling-edge retimer, and a path that skips the retimer cannot carry it. The bypass register is selected, captures 0 and leads the STAP chain, and TDO is retimed. A DR scan under ZERO_LENGTH_BYPASS is then one bit longer than the STAP chain and matches a DR scan under BYPASS bit for bit.

### TDO Retiming

The TDO output is retimed on the falling edge of TCK as required by IEEE 1149.1 Section 4.5.2. The retiming register ensures that TDO transitions occur at the same clock edge that samples TDI at the receiving device, maintaining timing margins across board-level JTAG chains. For zero-length bypass operation with the TAP_3DCR STAP-select bit clear, the retimer is bypassed using an explicit `prim_stdmux2` cell to minimize scan latency. While the bit is set, TDO is retimed under every instruction, ZERO_LENGTH_BYPASS included.

### ID Code

The IDCODE instruction accesses a TDR containing device identification information.

Table 2487 defines the identification fields shifted out by IDCODE.

*Table 2487. PTAP IDCODE fields*

| Bits | Field | Access | Description |
|---|---|---|---|
| 31:28 | si_rev | R | Silicon revision |
| 27:12 | part_num | R | Chiplet model ID |
| 11:1 | mfr_id | R | JEDEC-assigned manufacturer ID, encoded into 11 bits |
| 0 | marker | R | Marker bit, always 1 |

Bits shifted into the IDCODE TDR are ignored. The Programmer's Guide gives the TDR access sequence for this register.

### Debug Control

The DEBUG_CONTROL instruction accesses a TDR defining general control bits for debug and control of the design.

Table 2488 defines the clock-stop and debug-control fields of the DEBUG_CONTROL TDR.

*Table 2488. PTAP DEBUG_CONTROL fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 4 | cla_clock_stop | R | 0 | When set, indicates that a CLA in the chiplet attempted to stop the clocks. Note that the clocks will actually have been stopped only when *cla_clock_stop_en* is set. |
| 3 | jtag_clock_stop | RW | 0 | When set, stops all clocks in the chiplet. This is wired to the functional clock gaters on all of the PLLs in the chiplet. This bit is active high. |
| 2 | cla_clock_stop_en | RW | 0 | When set, enables the CLAs to stop the clocks in the design. This bit indicates that an external debugger is connected and ready to do debug or scan dump operations after the CLA stops the clocks. This bit is intended to protect against unintentional CLA clock stops in the field due to incorrect CLA programming. This bit is active high. |
| 1 | boot_stall_ovrd | RW | 0 | When set, overrides the boot stall signal to the internals of the part to use the value from boot_stall in this TDR instead. This bit is active high. |
| 0 | boot_stall | RW | 0 | When boot_stall_override is set, the value in this bit is used for boot stall. This bit is active high: setting to 1 causes boot stall. |

The DEBUG_CONTROL TDR captures bits shifted into its R/W fields and ignores those shifted into its RO fields. The Programmer's Guide gives the TDR access sequence for this register.

### JTAG Capabilities

The JTAG_CAPS instruction accesses a read-only TDR reporting the DTP version number and general configuration parameters for the DTP.

Table 2489 defines the version and configuration fields reported by JTAG_CAPS.

*Table 2489. PTAP JTAG_CAPS fields*

| Bits | Field | Access | Description |
|---|---|---|---|
| 59:54 | num_xtrig_int_ct | R | The number of internal cross trigger signals. |
| 53:48 | num_xtrig_ctp | R | The number of Cross Trigger Ports. |
| 47:44 | num_xtra_stap | R | The number of additional DTP STAP interfaces. |
| 43 | stap_io_en | R | The STAP interface for chiplet-to-chiplet connectivity is enabled. |
| 42 | sep_dbg_en | R | The SEP STAP interface enabled. |
| 41 | smc_dbg_en | R | The SMC JTAG2AXI interface is enabled. |
| 40:33 | num_smc_ic_rst | R | The number of SMC IC_RESET override ports. |
| 32:25 | num_sep_ic_rst | R | The number of SEP IC_RESET override ports. |
| 24:17 | num_ext_ic_rst | R | The number of external IC_RESET override ports. |
| 16 | ic_rst_inst_en | R | The IC_RESET instruction is enabled. |
| 15 | tmp_inst_en | R | The TMP controller functionality and instructions are enabled. |
| 14 | runbist_inst_en | R | The RUNBIST instruction is enabled. |
| 13 | highz_inst_en | R | The HIGHZ instruction is enabled. |
| 12 | clamp_inst_en | R | The CLAMP instruction is enabled. |

| Bits | Field | Access | Description |
|---|---|---|---|
| 11 | intest_inst_en | R | The INTEST instruction is enabled. |
| 10 | extest_pulse_en | R | The EXTEST_PULSE instruction is enabled. |
| 9 | extest_train_en | R | The EXTEST_TRAIN instruction is enabled. |
| 8 | bsr_inst_en | R | All mandatory boundary scan instructions are enabled. |
| 7:0 | och_ver | R | The MAJOR version number of the DTP IP. |

Bits shifted into the JTAG_CAPS TDR are ignored. The Programmer's Guide gives the TDR access sequence for this register.

#### Boundary Scan Support

All boundary scan and BIST instructions defined in IEEE 1149.1 are optionally supported, but support is not enabled by default. RTL parameters are provided to selectively enable support for each instruction. Any disabled instruction has the same behavior as the BYPASS instruction.

Any boundary scan instruction that selects the BSRs shifts their contents out on TDO. The Programmer's Guide gives the TDR access sequence for these registers.

#### BIST Support

The RUNBIST instruction is optionally supported, but support is not enabled by default. An RTL parameter is provided to selectively enable support. Any disabled instruction has the same behavior as the BYPASS instruction.

When the RUNBIST instruction is selected, the iJTAG network is selected as the active scan chain. An iJTAG network control signal indicates that the RUNBIST instruction is active. A designer may also utilize the decoded INST_SELECT signal to trigger the BIST logic for any instructions that they have enabled.

The PTAP adds no Segment Insertion Bit (SIB) or other scan cell to the iJTAG path. TDI is forwarded unchanged to the iJTAG network, and the network's return is the RUNBIST TDR, so the length and contents of the RUNBIST DR are defined entirely by the iJTAG network attached to the PTAP.

The Programmer's Guide gives the BIST execution sequence.

In the JTAG Interface Unit, the iJTAG network begins with three SIBs; their bit map and the integrated scan sequence are given in iJTAG Network.

#### Test Mode Persistence (TMP) Support

There is optional support for TMP, but support for TMP is disabled by default. An RTL parameter enables support for TMP.

When TMP support is enabled, a TMP controller is included in the PTAP. The TMP controller manages the active-low *CH_RESET_N* signal. The *CH_RESET_N* signal is the reset signal for any registers that must retain their state across TLR soft resets, mainly the boundary scan cells. *CH_RESET_N* asserts whenever the TAP is in the TLR state *except* when the TMP controller is in the *PERSISTENCE_ON* state. When the TMP controller is in the *PERSISTENCE_ON* state, the only events that will assert *CH_RESET_N* are full power-on-reset (POR) or TRST signal assertion.

When TMP support is disabled, the *CLAMP_HOLD*, *CLAMP_RELEASE*, and *TMP_STATUS* instructions have the same behavior as the *BYPASS* instruction, and the *CH_RESET_N* signal always responds to every POR, TRST, or TLR event, i.e. it is an exact copy of the *RESET_N* signal.

Table 2490 defines the bypass-escape and persistence-status bits.

*Table 2490. PTAP TMP_STATUS fields*

| Bit | Reset | Description |
|---|---|---|
| Bypass Escape (bit 0) | 0 | Whenever this bit is set, the TMP controller will transition to the *PERSISTENCE_OFF* state when the TAP controller enters the *UPDATE_IR* state after the *BYPASS* instruction has been selected. |
| TMP Status (bit 1) | 0 on capture | Capture-DR samples whether the TMP controller is in the Persistence-On state. The update register resets to `2'b10`; its bit 1 is replaced by the controller state on capture and does not enable persistence. |

### 3DIC Support

There is optional support for STAP interfaces, and support is enabled by default. An RTL parameter selects the number of STAP interfaces, where a value of 0 disables STAP support.

In order to properly support the IEEE 1838 standard, a chain of STAP interfaces must be inserted at the end of each chiplet's scan chain. This is done after the *IR Select* MUX to ensure that both the instruction registers and TDRs from all enabled PTAPs in the scan chain are scannable. As a consequence, the STAPs require dedicated capture, shift, and update control signals that are triggered during both the IR and DR sequences.

The PTAP has a 2-bit *3DCR* register that is accessed with the *TAP_3DCR* instruction. This register provides an option to bypass all STAPs in the scan chain.

Table 2491 defines the configuration-hold and STAP-selection bits.

*Table 2491. PTAP 3DCR fields*

| Bit | Reset | Description |
|---|---|---|
| Config Hold | 0 | When this bit is set, the PTAP TLR state does not reset this TDR, and it will hold its current state until TRST is asserted. |
| Select | 0 | When this bit is unset, the PTAP's top-level STAP scan interface is completely bypassed: TDO does not come from the STAP chain, and the select, capture, shift and update strobes of the STAP scan interface stay low, so the STAP chain holds its state through every IR and DR scan. |

The select bit takes effect at Update-DR of the scan that writes it. A scan that sets it therefore covers only this register, and the STAP chain joins the scan path from the next IR or DR scan onwards. Software sets the select bit in a TAP_3DCR scan of its own before programming any STAP.

When STAP support is disabled, the TAP_3DCR instruction has the same behavior as the *BYPASS* instruction, and the STAP scan chain cannot be selected.

### IC_RESET Support

The PTAP provides support for a configurable number of IEEE 1149.1-compliant reset controls. The IC_RESET TDR is composed of three independently-enabled slices:

- **SMC slice** (nearest TDI): driven by the `ic_reset_smc_t` struct type and gated by `IC_RESET_SMC_ENABLE`. Sized from `$bits(ic_reset_smc_t)`.
- **SEP slice**: driven by the `ic_reset_sep_t` struct type and gated by `IC_RESET_SEP_ENABLE`. Sized from `$bits(ic_reset_sep_t)`.
- **External slice** (nearest TDO): driven by the `ic_reset_ext_t` struct type and gated by `IC_RESET_EXT_ENABLE`. Sized from

`$bits(ic_reset_ext_t)`. The instantiator supplies this type; the PTAP does not name the external ports.

Each slice's struct must contain `.ovrd` and `.val` sub-structs of identical width (checked via an elaboration-time assertion in `jtag_ptap`). Within `jtag_ic_reset_reg`, the reset-enable and reset-control bits are interleaved in IEEE 1149.1 §17 order: in each port's pair, `reset_enable` is nearer TDO and `reset_control` nearer TDI, so port *n* occupies TDR bits `2*n+1` and `2*n+2` (Table 2492). `jtag_ptap` packs the three struct slices into that interleaved bus and unpacks the outputs back into structured form. The TDR carries the SMC slice closest to TDI, then SEP, then the external slice, followed by the single `reset_hold` bit.

The total length is `2 * (NumSmcIcReset + NumSepIcReset + NumExtIcReset) + 1` bits. Each `Num*IcReset` is half the width of its slice's struct type, `$bits(ic_reset_*_t) / 2`, when the matching `IC_RESET_*_ENABLE` parameter is set, and 0 otherwise.

Table 2492 defines the reset-control, reset-enable, and bypass fields in the IC_RESET TDR.

*Table 2492. PTAP IC_RESET fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 2*n+2 | reset_control | RW | 1 | The controlled, active-low reset value driven while *reset_enable* is 0 (i.e. while JTAG override is active). |
| 2*n+1 | reset_enable | RW | 1 | Active-LOW per IEEE 1149.1 §17: 1 (default) ⇒ JTAG override DISABLED (upstream reset passes through); 0 ⇒ JTAG override ENABLED (the port is driven by *reset_control*). The "enable" in the IEEE name refers to "enabling the controlled reset state", not the override mechanism. |
| 0 | reset_hold | RW | 1 | When 0, TLR will **not** reset the *reset_enable* and *reset_control* bits. TRST/POR will always reset all bits. |

**IMPORTANT**

**Polarity convention** — the TDR field names come from the IEEE 1149.1 §17 specification and use an active-low `reset_enable`. `jtag_ic_reset_reg` converts this into a conventional **active-high** override signal (`ic_reset_ovrd_o[i] = !reset_enable[i]`) before it leaves the register, so every downstream consumer sees `.ovrd == 1` meaning "JTAG is currently overriding this port". Do not re-invert this signal downstream.

For each of the resets defined in the TDR, the PTAP exposes two logical signals (grouped into the per-slice `jtag_*_reset_ctrl_t` structs):

- *ic_reset_ovrd* / `.ovrd` - **Active-high** JTAG-override flag (the bit-wise inversion of *reset_enable*). 1 ⇒ JTAG is driving this reset, 0 ⇒ upstream reset passes through.
- *ic_reset_ctrl_n* / `.val` - Active-low reset value driven when *ic_reset_ovrd* is asserted (identical to the *reset_control* TDR bit).

#### iJTAG Support

The PTAP provides a scan port for an IEEE 1687-2014-compliant hierarchical iJTAG network. The *SELECT_IJTAG* instruction is always decoded; no configuration parameter removes it. While it is the current instruction, a DR scan selects the iJTAG network and the network return drives TDO.

#### JTAG2AXI Support

#### *_JTAG2AXI_CAPS

This is a private instruction that accesses a read-only TDR reporting the configuration parameters for the given JTAG2AXI debugging instrument.

Table 2493 defines the configuration fields reported by each JTAG2AXI instrument.

*Table 2493. PTAP JTAG2AXI capability fields*

| Bits | Field | Access | Description |
|---|---|---|---|
| 13:12 | rd_pl_depth | R | The depth of the transaction read queue where 0 indicates only a single outstanding transaction allowed at any given time. |
| 11:10 | wr_pl_depth | R | The depth of the transaction write queue where 0 indicates only a single outstanding transaction allowed at any given time. |
| 9:7 | data_size | R | The size of AxDATA in powers-of-2 bytes. 0: 1 byte (8 bits) 1: 2 bytes (16 bits) 2: 4 bytes (32 bits) 3: 8 bytes (64 bits) 4: 16 bytes (128 bits) 5: 32 bytes (256 bits) 6: 64 bytes (512 bits) 7: 128 bytes (1024 bits) |
| 6:1 | addr_size | R | The size of AxADDR in bits. |
| 0 | bus_type | R | The type of the AXI bus. 0: AXI4 1: AXI4-Lite |

***_AXI_SINGLE_OP**

This is a private instruction that generates a single AXI read or write transaction. It provides a mechanism for an external debugger to do sparse reads or writes to nearly any register in the design. Since the entire address must be shifted in for each transaction, this instruction is most useful for sparse accesses. This instruction uses semantics very similar to the RISC-V DTM DMI instruction.

Let `D` be the AXI data width in bits, `A` the address width, and `S` the scan size-field width: 1 bit for `D ⇐ 16`, 2 bits for `D ⇐ 64`, and 3 bits otherwise. The single-operation TDR has `2 + S + D/8 + D + A` bits. For `D=64` and `A=51`, it has 127 bits.

The size field requests a transfer of `2**size` bytes. The AXI protocol does not allow a transfer wider than the data bus, so the bridge uses the effective size `E = min(size, data_size)`, where `data_size = log2(D/8)` is the capability field of the same name. A size above `data_size` transfers one full `D`-bit beat: for `D=32`, the 2-bit size field admits 3, and the bridge transfers 4 bytes.

Figure 85 shows the single-operation TDR for a 64-bit data bus and a 51-bit address bus. In this and the following layout figures the most significant bit is on the left, nearest TDI, and bit 0 on the right shifts out to TDO first. The bit range of each field is above it and its width below it; field widths are not drawn to scale.

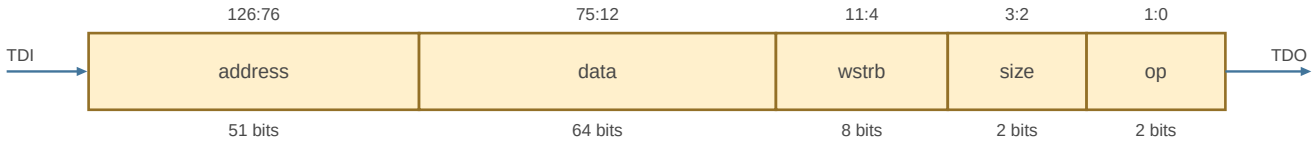


*Figure 85. JTAG2AXI single-operation layout*

Table 2494 defines the address, data, byte-enable, size, and operation fields.

*Table 2494. JTAG2AXI single-operation fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| (2+S+D/8+D) +: A | address | RW | 0 | Address used for AXI access. In Update-DR the AXI transaction is issued. On a Capture-DR, this field holds the previous value. |
| (2+S+D/8) +: D | data | RW | 0 | The data to write on an AXI write during Update-DR. On a Capture-DR, this field contains the data returned from an AXI read if the previous operation was a read, or holds the write data previously used if the previous operation was a write. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| (2+S) +: (D/8) | wstrb | RW | 0 | The byte strobes for a write transaction. This value is driven on WSTRB when the transaction is a write. On a Capture-DR, this field holds the previous value. |
| 2 +: S | size | RW | 0 | The requested transfer size in powers-of-2 bytes. The effective size `E`, the smaller of this field and `data_size`, is used as AxSIZE for the transaction. 0: 1 byte (8 bits) 1: 2 bytes (16 bits) 2: 4 bytes (32 bits) 3: 8 bytes (64 bits) 4: 16 bytes (128 bits) 5: 32 bytes (256 bits) 6: 64 bytes (512 bits) 7: 128 bytes (1024 bits) |
| 1:0 | op | RW | 0 | When the debugger writes this field, it has the following meaning: 0: Ignore data and address (nop) Don't send anything over AXI during Update-DR. This operation should never result in a busy or error response. 1: Read from address (read) 2: Write data to address (write) 3: Reserved When the debugger reads this field, it means the following: 0: The previous operation completed successfully and returned an OKAY status 1: The previous operation completed but failed and returned a SLVERR status 2: The previous operation completed but failed and returned a DECERR or other error status, or a system reset discarded it (see System Reset Recovery). 3: The latest operation was attempted before the previous operation completed; the previous AXI request is still in progress. The operation attempted in this access will be ignored. If a debugger sees this status, it needs to give the target more TCK edges between Update-DR and Capture-DR. The simplest way to do that is to add extra transitions in Run-Test/Idle. |

***_AXI_SERIES_CTRL**

This is a private instruction that preloads the start address of a region that will be subsequently read or written using the AXI_SERIES_DATA instruction. It is used when an external debugger wants to either:

- Read or write a contiguous block of registers or memory.
- Repeatedly read or write the same location (e.g., polling)

Using the series instructions instead of the AXI_SINGLE_OP instruction is much more time efficient for these use cases.

The pipeline-depth field width `P` is 1 when `FIFO_DEPTH=0`, otherwise `$clog2(FIFO_DEPTH+1)`. The control TDR has `2 + S + P + A + 1` bits. For `D=64`, `A=51`, and `FIFO_DEPTH=3`, the length is 58 bits.

Figure 86 shows the series-control TDR for 64-bit data, 51-bit addresses, and FIFO_DEPTH=3.

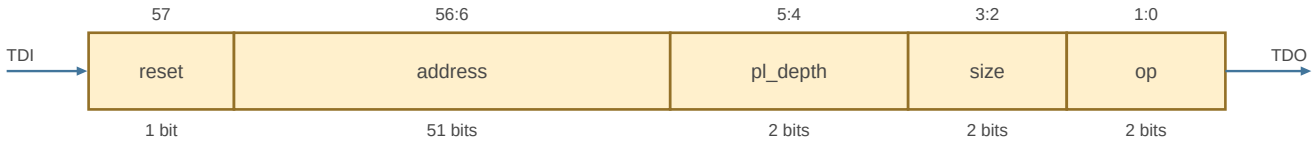


*Figure 86. JTAG2AXI series-control layout*

Table 2495 defines the control fields used to start and manage a series of AXI accesses.

*Table 2495. JTAG2AXI series-control fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| (2+S+P+A) | reset | RW | 0 | Reset the pending error flags reported in op. |
| (2+S+P) +: A | address | RW | 0 | Address used for AXI access. In Update-DR the AXI transaction is issued. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| (2+S) +: P | pl_depth | RW | 0 | pipeline_depth For write transactions, this sets the number of pending unissued write transactions that can be queued by this block to pl_depth + 1. Setting a non-zero value allows the debugger to queue additional write transactions before the previous ones have been issued. For read transactions, this field sets the number of read responses that this block will pipeline. This allows the debugger to queue multiple reads to targets with long or variable latency while getting read data responses in a deterministic fashion. The first pl_depth + 1 reads will return DEADBEEF marker data. This field can be set to a value between 0 and the (wr/rd)_pl_depth field read from the bridge's `*_JTAG2AXI_CAPS` register. When the debugger reads this field, an indication of whether there are any pending transactions will be returned: 2'b00: There are no pending transactions 2'b01: There is at least one pending transaction |
| 2 +: S | size | RW | 0 | The requested transfer size in powers-of-2 bytes. The effective size `E`, the smaller of this field and `data_size`, is used as AxSIZE for the transaction. 0: 1 byte (8 bits) 1: 2 bytes (16 bits) 2: 4 bytes (32 bits) 3: 8 bytes (64 bits) 4: 16 bytes (128 bits) 5: 32 bytes (256 bits) 6: 64 bytes (512 bits) 7: 128 bytes (1024 bits) |
| 1:0 | op | RW | 0 | When the debugger writes this field, it has the following meaning: 0: Ignore data and address (nop). Don't do anything on Update-DR for AXI_SERIES_DATA. 1: Read from address (read). Update-DR for AXI_SERIES_DATA causes an AXI read to be performed. 2: Write data to address (write). Update-DR for AXI_SERIES_DATA causes an AXI write to be performed. 3: Reserved; treated as nop. When the debugger reads this field, it means the following: 0: All previous series operations completed successfully and returned an OKAY status. 1: At least one previous series operation completed but failed and returned a SLVERR status. This status is sticky until cleared by AXI_SERIES_CTRL.reset. 2: At least one previous series operation completed but failed and returned a DECERR or other error status, or a system reset discarded a series operation that was in progress or queued (see System Reset Recovery). This status is sticky until cleared by AXI_SERIES_CTRL.reset. Codes 1 and 2 report the first failed series operation since the last AXI_SERIES_CTRL.reset; a later failure does not replace it. The completion status of an AXI_SINGLE_OP operation is reported only through AXI_SINGLE_OP.op and does not change codes 0 to 2. 3: At least one operation was attempted before the previous operation completed. The operations attempted when there was a pending operation in progress were ignored. If a debugger sees this status, it needs to give the target more TCK edges between Update-DR and Capture-DR. The simplest way to do that is to add extra transitions in Run-Test/Idle. This status is sticky until cleared by AXI_SERIES_CTRL.reset. |

***_AXI_SERIES_DATA_INCR**

This is a private instruction that does the actual read/write to the address previously specified.

After the access is completed, an internal address register used for the series, which was previously loaded by AXI_SERIES_CTRL, is incremented as follows:

AXI_SERIES_CTRL.address += 2**E

This allows the debugger to load or dump contiguous regions of address space.

Note that this is a variable-length data register. The length is n bits where n = 8*(2**E).

The data field is the logical transfer payload. The bridge places that payload on the AXI byte lanes selected by the current address and the effective size `E`, and Capture-DR returns the payload from those lanes.

The data field is drawn as the payload bytes it holds, with bit 0 aligned across the four sizes. The incrementing payload

layouts for effective sizes E of 0, 1, 2, and 3 are shown in Figure 87, Figure 88, Figure 89, Figure 90, respectively. Effective size 3 needs a 64-bit data bus, which the last figure assumes.

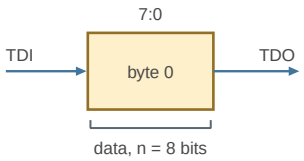


*Figure 87. JTAG2AXI incrementing data layout for effective size 0*

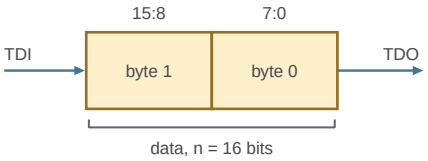


*Figure 88. JTAG2AXI incrementing data layout for effective size 1*

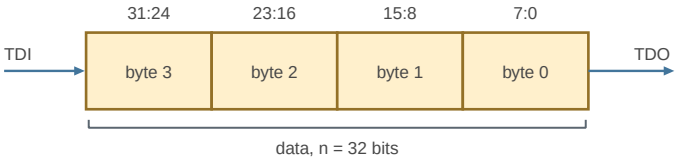


*Figure 89. JTAG2AXI incrementing data layout for effective size 2*

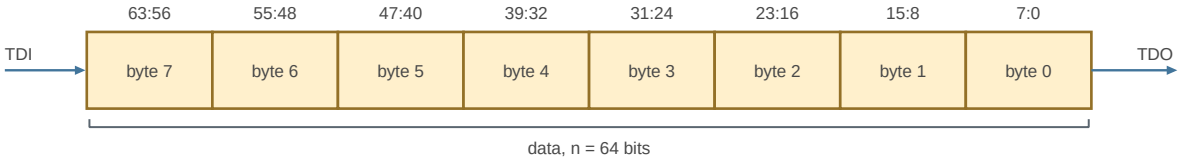


*Figure 90. JTAG2AXI incrementing data layout for effective size 3*

Table 2496 defines the payload shifted through the incrementing data instruction.

*Table 2496. JTAG2AXI incrementing data fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| n-1:0 | data | RW | 0 | Data value to read/write. The read data is from the previous read access. |

***_AXI_SERIES_DATA_NO_INCR**

This is a private instruction that does the actual read/write to the address previously specified.

Unlike AXI_SERIES_DATA_INCR, this instruction does not change the address in AXI_SERIES_CTRL. This instruction allows the debugger to continue to read or write this same address using additional AXI_SERIES_DATA_NO_INCR instructions. This is useful for repeated polling or triggering.

Note that this is a variable-length data register. The length is n bits where n = 8*(2**E).

The data field is the logical transfer payload. The bridge places that payload on the AXI byte lanes selected by the current address and the effective size E, and Capture-DR returns the payload from those lanes.

The non-incrementing data register has the same layout as the incrementing one, so the same figures apply. The non-incrementing payload layouts for effective sizes E of 0, 1, 2, and 3 are shown in Figure 91, Figure 92, Figure 93, Figure 94, respectively.

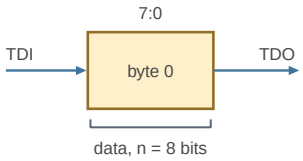


*Figure 91. JTAG2AXI non-incrementing data layout for effective size 0*

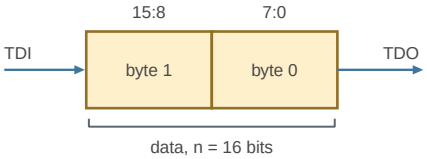


*Figure 92. JTAG2AXI non-incrementing data layout for effective size 1*

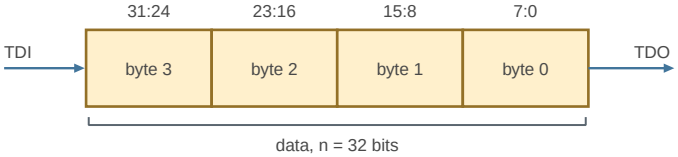


*Figure 93. JTAG2AXI non-incrementing data layout for effective size 2*

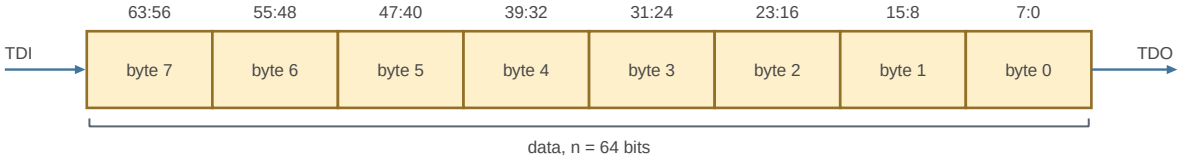


*Figure 94. JTAG2AXI non-incrementing data layout for effective size 3*

Table 2497 defines the payload shifted through the non-incrementing data instruction.

*Table 2497. JTAG2AXI non-incrementing data fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| n-1:0 | data | RW | 0 | Data value to read/write. The read data is from the previous read access. |

***_AXI_SERIES_DATA_WITH_ERROR_STATUS**

This is a private instruction that does the actual read/write to the address previously specified.

Unlike AXI_SERIES_DATA_INCR and AXI_SERIES_DATA_NO_INCR, the increment is selectable per transaction and the error status from the previous status is returned. This instruction is used when granular error reporting is desired during a block operation.

Note that this is a variable-length data register. The length is n+1 bits where n = 8*(2**E).

The data field is the logical transfer payload. The bridge places that payload on the AXI byte lanes selected by the current address and the effective size E, and Capture-DR returns the payload from those lanes.

The error-status payload layouts for effective sizes E of 0, 1, 2, and 3 are shown in Figure 95, Figure 96, Figure 97, Figure 98, respectively. Effective size 3 needs a 64-bit data bus, which the last figure assumes.

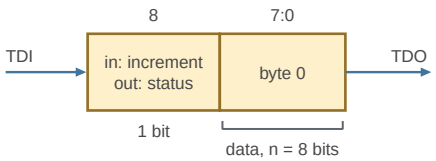


*Figure 95. JTAG2AXI error-status data layout for effective size 0*

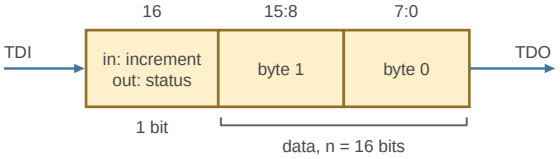


*Figure 96. JTAG2AXI error-status data layout for effective size 1*

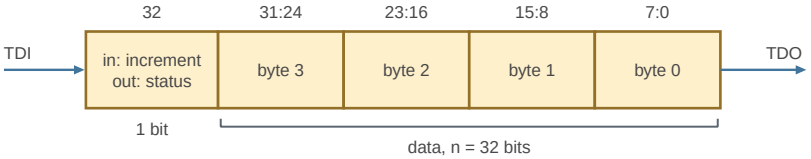


*Figure 97. JTAG2AXI error-status data layout for effective size 2*

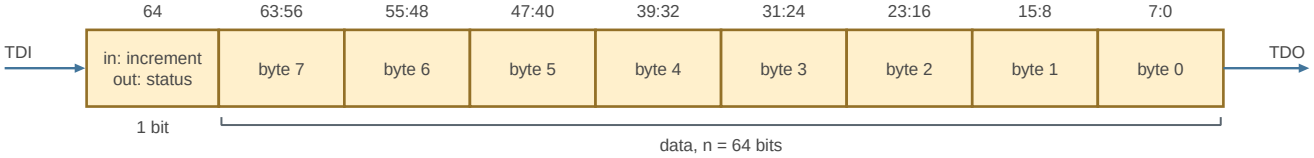


*Figure 98. JTAG2AXI error-status data layout for effective size 3*

Table 2498 defines the data payload and the shared increment-control and error-status bit.

*Table 2498. JTAG2AXI data and error-status fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| n | increment/status | RW | 0 | The value shifted into this bit indicates whether the address should be incremented after this transaction completes. The value shifted out of this bit indicates whether the previous transaction completed successfully (0 = succeeded, 1 = failed). An AXI result of SLVERR or DECERR, a system reset that discarded the transaction, as well as attempting a new transaction while the previous was still pending will cause the status to be returned as "failed". |
| n-1:0 | data | RW | 0 | Data value to read/write. The read data is from the previous read access. |

### System Reset Recovery

A system reset resets the AXI side of each JTAG2AXI instrument while its JTAG side keeps running. The instrument recovers without a TAP reset. When the reset reaches its JTAG side, it discards every operation that is in flight or queued: the AXI transaction it was waiting on, series requests not yet issued, and series read data not yet shifted out. The target can still complete a transaction that was already on the AXI bus before the reset reached it, so the debugger must treat a discarded write as having an unknown effect.

After the discard:

- The SINGLE_OP op field reads 2 if a single operation or an error-status series operation was discarded, and its captured read-data field reads 0. Otherwise both keep their values.
- The SERIES_CTRL op field reads 2 if a series operation was discarded and it held no earlier series error. Otherwise it keeps its value, including a sticky status of 3. Only AXI_SERIES_CTRL.reset clears it.
- The SERIES_CTRL address, size, pl_depth and op settings keep their values, but the read pipeline restarts empty. Reprogram SERIES_CTRL before resuming a pipelined series read.

Recovery starts when the system reset asserts and completes within 52 periods of the slower of TCK and the AXI clock after the system reset deasserts. TCK periods count only while the debugger drives TCK, for example with transitions in Run-Test/Idle. An operation shifted in during that time is held and issued once recovery completes, and the SINGLE_OP op field reads 3 until the operation finishes.

### Debug Disable

The PTAP has three per-bridge debug-disable inputs. The JTAG interface unit synchronizes those bits to TCK and presents them here as ready-to-gate signals. Each input is active-high: when it is 1, the corresponding `jtag2axi` instance blocks AXI transaction generation.

Table 2499 maps the three JTAG2AXI bridges to their active-high disable inputs.

*Table 2499. PTAP debug-disable inputs*

| Debug Interface | Disable Input |
|---|---|
| SMC Fabric JTAG2AXI | `smc_jtag2axi_security_disable_i` |
| SMC OTP JTAG2AXIL | `smc_otp_jtag2axi_security_disable_i` |
| SEP OTP JTAG2AXIL | `sep_otp_jtag2axi_security_disable_i` |

### Interface

### Instruction Encodings

The PTAP uses 6-bit instruction encodings. During the capture phase, the least significant bit (closest to TDO) of the

instruction register always presents a 1 to the scan chain, and the next least significant bit presents a 0 as specified by the 1149.1 standard, section 7.2.1. All other bits of the instruction register always return 0s.

Table 2500 lists the PTAP instruction encodings and associated register lengths.

*Table 2500. PTAP instruction encodings*

| Encoding | Name | Register Size | Description |
|---|---|---|---|
| 0x00 | BYPASS | 1 | IEEE 1149.1, Section 8.4 |
| 0x01 | IDCODE | 32 | IEEE 1149.1, Section 8.13 |
| 0x02 | RUNBIST | BIST | IEEE 1149.1, Section 8.10 |
| 0x03 | SAMPLE_PRELOAD | BSR | IEEE 1149.1, Sections 8.6 and 8.7 |
| 0x04 | EXTEST | BSR | IEEE 1149.1, Section 8.8 |
| 0x05 | EXTEST_TRAIN | BSR | |
| 0x06 | EXTEST_PULSE | BSR | |
| 0x07 | CLAMP | 1 | IEEE 1149.1, Section 8.11 |
| 0x08 | HIGHZ | 1 | IEEE 1149.1, Section 8.16 |
| 0x09 | INTEST | BSR | IEEE 1149.1, Section 8.9 |
| 0x0A | CLAMP_HOLD | 2 | IEEE 1149.1, Section 8.20 |
| 0x0B | CLAMP_RELEASE | 2 | IEEE 1149.1, Section 8.20 |
| 0x0C | TMP_STATUS | 2 | IEEE 1149.1, Section 8.20 |
| 0x0D | IC_RESET | Variable | IEEE 1149.1, Section 8.21 |
| 0x0E | TAP_3DCR | 2 | IEEE 1838, Section 5.5.1 |
| 0x18 | DEBUG_CONTROL | 5 | Debug and clock stop control |
| 0x19 | JTAG_CAPS | 60 | Capabilities readback |
| 0x1A | SELECT_IJTAG | Variable | iJTAG network selection |
| 0x1B | SMC_OTP_JTAG2AXI_CAPS | 14 | SMC OTP bridge capabilities readback |
| 0x1C | SMC_OTP_AXI_SINGLE_OP | Variable | SMC OTP bridge single transaction |
| 0x1D | SMC_OTP_AXI_SERIES_CTRL | Variable | SMC OTP bridge series control |
| 0x1E | SMC_OTP_AXI_SERIES_DATA_INCR | Variable | SMC OTP bridge series data, incrementing address |
| 0x1F | SMC_OTP_AXI_SERIES_DATA_NO_INCR | Variable | SMC OTP bridge series data, fixed address |
| 0x20 | SMC_OTP_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | SMC OTP bridge series data with error status |
| 0x21 | SEP_OTP_JTAG2AXI_CAPS | 14 | SEP OTP bridge capabilities readback |
| 0x22 | SEP_OTP_AXI_SINGLE_OP | Variable | SEP OTP bridge single transaction |
| 0x23 | SEP_OTP_AXI_SERIES_CTRL | Variable | SEP OTP bridge series control |
| 0x24 | SEP_OTP_AXI_SERIES_DATA_INCR | Variable | SEP OTP bridge series data, incrementing address |
| 0x25 | SEP_OTP_AXI_SERIES_DATA_NO_INCR | Variable | SEP OTP bridge series data, fixed address |
| 0x26 | SEP_OTP_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | SEP OTP bridge series data with error status |
| 0x27 | SMC_JTAG2AXI_CAPS | 14 | SMC fabric bridge capabilities readback |

| Encoding | Name | Register Size | Description |
|---|---|---|---|
| 0x28 | SMC_AXI_SINGLE_OP | Variable | SMC fabric bridge single transaction |
| 0x29 | SMC_AXI_SERIES_CTRL | Variable | SMC fabric bridge series control |
| 0x2A | SMC_AXI_SERIES_DATA_INCR | Variable | SMC fabric bridge series data, incrementing address |
| 0x2B | SMC_AXI_SERIES_DATA_NO_INCR | Variable | SMC fabric bridge series data, fixed address |
| 0x2C | SMC_AXI_SERIES_DATA_WITH_ERROR_STATUS | Variable | SMC fabric bridge series data with error status |
| 0x3D | ZERO_LENGTH_BYPASS | 0, or 1 | TDI to TDO with no register; BYPASS, with its 1-bit register, while the TAP_3DCR STAP-select bit is set |
| 0x3E | INV_BYPASS | | |
| 0x3F | BYPASS | 1 | IEEE 1149.1, Section 8.4 |

The length of each variable-length JTAG2AXI register follows from the bridge's address width, data width and pipeline depth, and for the series data registers from the size last written to `*_AXI_SERIES_CTRL`. JTAG2AXI Support defines each layout.

**NOTE:** All instruction encodings *not* explicitly defined in this table map to the BYPASS instruction.

### Signal Interface

Packed control and bus types remain bundles at the `jtag_ptap` boundary; their members are not separate scalar ports.

The parameters and ports of `jtag_ptap` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Dependencies

The PTAP depends on the following components:

- `hw/common/ocah_prim` — Primitive cells (synchronizers, clock gating)
- `hw/ip/jtag/jtag_stap` — Secondary TAP (when 3DIC support enabled)

### JTAG STAP

The Secondary TAP (STAP) enables hierarchical interconnection of TAP controllers across multiple chiplets as specified in IEEE 1838-2019 (3DIC). Each STAP interface extends the SiP's primary JTAG access across internal or external chiplets.

### Overview

The DTP supports these STAP types:

- **I/O STAP** — extends JTAG to the next chiplet in the SiP.
- **SMC Debug STAP** — provides access to the SMC CPU core debugger TAP.
- **SEP Debug STAP** — provides access to the SEP CPU core debugger TAP.
- **Additional STAPs** — optional connections to additional on- or off-chip TAPs.
- **Extended STAP Scan Interface** — a port for adding STAPs external to the DTP.

Each STAP has an active-high disable input. When it is asserted, STAP selection, host TDO, and 3DCR updates are

blocked.

### Configuration Parameters

Three timing parameters are available to optimise the STAP for different physical topologies, particularly die-boundary crossings.

The parameters of `jtag_stap` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Architecture

IEEE 1838-2019 enables hierarchical interconnection of TAP controllers across multiple chiplets. Each STAP interface provides an extension of a SiP's primary JTAG interface across internal chiplets. When enabled, a STAP interface is part of the JTAG scan chain, otherwise the STAP interface is bypassed. In addition, STAPs can also support connection of other TAP interfaces included on the same chiplet as the PTAP.

### Module Hierarchy

The STAP implementation consists of three main components integrated within a single module (Table 2501).

*Table 2501. STAP module hierarchy*

| Module | Purpose | Conditional |
|---|---|---|
| `prim_jtag_sib_mux_pre` | Segment Insertion Bit multiplexer for TAP control signal routing | Always |
| `prim_jtag_scan_reg` | 3-bit 3DCR control register (config_hold, stap_sel, tms_hold) | Always |
| Lockup latches | TDO lockup (always), TDI lockup (optional), scan output lockup (optional) | Per parameter |

### STAP Module

Each STAP has a 3-bit 3DCR TDR that defines the control interface for the STAP. Note that the STAP 3DCRs take a distributed segment approach to improve timing, as opposed to the single monolithic PTAP 3DCR approach detailed in the IEEE 1838 standard. This 3DCR is connected inline within each STAP's scan chain. A SIB is inserted at the end of the 3DCR to exclude it from the scan chain when not needed. The one-bit SIB is always in the scan chain, so a STAP adds one bit to the chain while its SIB is closed and four bits while it is open. While the STAP is selected, the downstream TAP's scan path is also in the chain, between the STAP's scan input and the SIB.

When not selected, the STAP interface TMS signal is tied to the constant value specified by the TMS-Hold bit of the 3DCR, otherwise it follows the PTAP TMS signal.

Figure 99 shows the scan path, the host TDO path and the gating logic of one STAP. Blue lines carry scan data and grey lines carry control. A small circle marks an inverted gate input, and a name at the tail of a short arrow (`stap_sel`, `security_disable_i` and `client_scan_ctrl_i.shift_en`) is the same signal drawn elsewhere in the figure. `stap_scan_in` feeds both the scan-in mux and the TDO lockup flop, whose output leaves on `host_tdo_o` toward the secondary TAP. While `stap_sel` is set, the scan-in mux passes the secondary TAP's return on `host_tdi_i`, which splices that TAP into the chain; otherwise it passes `stap_scan_in`. The SIB inserts the 3DCR into the client scan path or bypasses it, and its one-bit register drives `client_scan_out_o`. The dashed input stages are optional: SCAN_IN_PIPE on `client_scan_in_i` defaults to disabled for all DTP instances, and TDI_LOCKUP on `host_tdi_i` defaults to disabled but is enabled for the I/O STAP (`gen_stap_io`) to provide timing margin for the die boundary crossing.

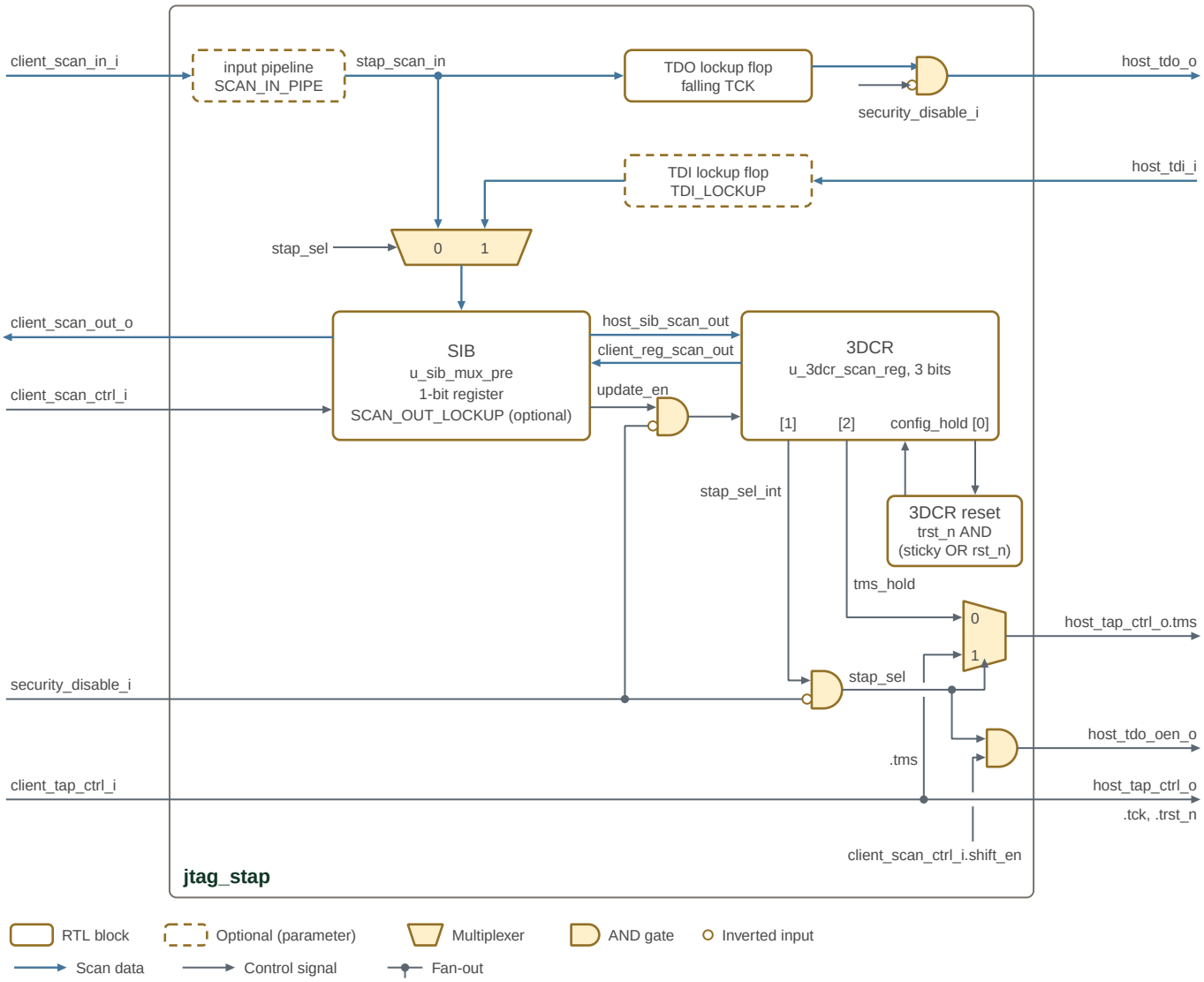


*Figure 99. STAP scan and control paths*

`security_disable_i` gates three signals through AND gates with an inverted input: `stap_sel`, from the 3DCR `stap_sel_int` bit; the 3DCR update enable, from the SIB's `update_en`; and `host_tdo_o`. Because the TMS mux and the TDO output enable take the gated `stap_sel`, a disabled STAP also parks the host TMS at `tms_hold` and holds `host_tdo_oen_o` low. `host_tap_ctrl_o` carries TCK and TRST from `client_tap_ctrl_i` unchanged; only its TMS passes through the mux. Where the TCK and TRST line crosses the shift-enable input, the gap in the shift-enable line marks that the two do not connect. The SIB itself is not gated. The 3DCR reset combines the client TRST, `config_hold_sticky` and the scan reset as Table 2502 describes.

### 3DCR Register

The 3-bit 3DCR control register is implemented using `prim_jtag_scan_reg` with a loopback configuration. The register outputs feed back to the data inputs, allowing the register to retain its configuration across scan operations while still being scannable for updates.

Table 2502 defines the three control bits retained in the STAP scan register.

*Table 2502. STAP 3DCR fields*

| Bit | Field | Reset | Description |
|---|---|---|---|
| 2 | TMS-Hold | 0 | Defines the parked TMS value when the STAP is disabled. When `stap_sel=0`, the host TMS output is driven with this static value rather than following the client TMS input. |
| 1 | Select (stap_sel) | 0 | Enables the STAP scan interface. When `stap_sel=1`, the STAP routes scan data through the downstream TAP. When `stap_sel=0`, the downstream TAP is not in the scan path and its TMS is held at the TMS-Hold value. While `security_disable_i` is high, a capture of the 3DCR reads this bit as 0 whatever its stored value (Table 2504). |
| 0 | Config-Hold | 0 | Preserves the 3DCR configuration across Test-Logic-Reset. The active-low register reset is `client_tap_ctrl_i.trst_n AND (config_hold_sticky OR host_sib_scan_ctrl.rst_n)`, where `config_hold_sticky` samples `config_hold` on the falling edge of TCK and resets only on TRST. TRST assertion always resets the register regardless of this bit. |

### STAP Types

Table 2503 lists the STAPs allocated for the DTP JTAG interface. The STAPs are serially chained TDI → TDO according to their order in this table from top to bottom. In addition to the STAPs integrated into the DTP, the STAP scan chain is ported out of the DTP so that the chiplet integrator may add additional STAPs closer to their point of use.

*Table 2503. DTP STAP allocation*

| Purpose | Description |
|---|---|
| STAP to Next Chiplet | This STAP forwards JTAG to the next chiplet in the SiP. |
| SMC Core Debug | Provides access to the internal SMC CPU core debugger TAP. |
| SEP Core Debug | Provides access to the internal SEP CPU core debugger TAP. |
| [N-1:0] Additional STAPs | Provides access to additional STAPs for on or off-chip TAP connectivity. |
| Extended STAP Scan Interface | Provides the ability to add STAPs external to the DTP closer to their point of use. |

### Debug Disable

The `security_disable_i` input is active-high (1 = STAP disabled). When it is asserted, three aspects of STAP operation are blocked (Table 2504).

*Table 2504. STAP debug-disable behavior*

| Signal | Behavior when asserted |
|---|---|
| `stap_sel` (internal) | Forced to 0 regardless of the 3DCR Select bit, so scan data is not routed through the downstream TAP. The 3DCR captures this forced value, so a scan through the 3DCR reads Select as 0. |
| `host_tdo_o` | Forced to 0, so the downstream TAP cannot leak data on the host TDO even if it is accessed. |
| `client_reg_scan_ctrl.update_en` | Forced to 0, so the 3DCR cannot be updated while the STAP is disabled. The stored Select bit keeps its value: when `security_disable_i` clears, a STAP whose stored Select bit is 1 is selected again without a 3DCR write. The disable does not block the 3DCR reset, so TRST clears the stored bit, and so does the scan reset while Config-Hold is 0. |

STAP selection, host TDO, and 3DCR updates are all blocked combinationally at the module output, so there is no timing-dependent bypass of the disable.

### Lockup Latches

The STAP implements three types of lockup latches to manage timing across die boundaries and long scan chains. These latches are clocked on the falling edge of TCK to capture data during the second half of the clock cycle, providing setup time margin for the next rising edge.

#### TDO Lockup Latch

The TDO lockup latch is always present and cannot be disabled. On every falling edge of TCK it captures the client scan data entering the STAP and drives it on `host_tdo_o` to the downstream TAP's TDI. The downstream TAP samples TDI on the rising edge of TCK, so the latch gives that path half a TCK period of setup margin.

#### TDI Lockup Latch

The TDI lockup latch is optional and controlled by the `TDI_LOCKUP` parameter (default = 0). When enabled, this latch captures the host TDI input on the falling edge of TCK before routing it to the client TDI output. This adds one additional cycle of latency but provides timing margin for die boundary crossings where signal delays may be significant. In the DTP integration, TDI lockup is enabled only for the I/O STAP that crosses between chiplets.

#### Scan Output Lockup Latch

The scan output lockup latch is optional and controlled by the `SCAN_OUT_LOCKUP` parameter (default = 0). This latch is implemented within the `prim_jtag_sib_mux_pre` module and retimes the client scan output on the falling edge of TCK. Enable it where the client scan output feeds a rising-edge stage clocked by a later TCK branch. Leave it disabled where the output reaches a falling-edge stage, such as the PTAP TDO retimer: two falling-edge stages in series add one TCK of scan latency. The DTP integration leaves it disabled on every STAP.

### Interface

#### Signal Interface

Packed control and bus types remain bundles at the `jtag_stap` boundary; their members are not separate scalar ports.

The parameters and ports of `jtag_stap` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Timing Requirements

The STAP interface timing follows IEEE 1838-2019:

- TDI is sampled on the rising edge of TCK
- TDO is driven on the falling edge of TCK
- TMS is sampled on the rising edge of TCK
- The TDO lockup latch captures the client scan data on every falling edge of TCK

### Dependencies

The STAP depends on the following components:

- `hw/common/ocah_prim` - Primitive cells (clock gating, synchronizers)

### Cross-Trigger

#### Cross-Trigger Network

The Cross-Trigger Network (CTN) implements inter-chiplet and on-chip cross-trigger routing per the Open Chiplet Cross Triggering (OCCT) specification. It combines a configurable set of Cross Trigger Ports (CTPs) facing the I/O pads with a Cross Trigger Matrix (CTM) that routes trigger pulses between all sources and sinks under AXI-Lite CSR control.

### Overview

The CTN serves four purposes:

1. **Die-to-die cross triggering** — trigger pulses propagate between chiplets in a multi-die SiP through the CTP GPIO pads.
2. **On-chip cross triggering** — internal CLAs and other logic exchange trigger pulses without leaving the chiplet.
3. **Unified management** — a single AXI-Lite interface configures all CTP and CTM settings through a contiguous address window.
4. **Clock stop coordination** — the CTN OR-reduces CLA `xtrig_clk_stop_req_i` inputs and combines the result with the JTAG `jtag_clock_stop` bit to produce the registered `stop_clks_o` output.

### Configuration Parameters

The CTN is configured through its external and internal port counts, the number of clock-stop request inputs, and the protocol of each internal CTP.

The parameters of `cross_trigger_network` are listed in the HTML edition of this manual, under RTL Modules Reference.

The DTP passes these counts through as `XTRIG_NUM_CTP`, `XTRIG_NUM_INT_CT`, and `XTRIG_NUM_CLK_STOP_REQ`. The elaborated `NUM_CTP + NUM_INT_CT` must equal the generated Cross-Trigger Matrix `NUM_CT_SRC` and `NUM_CT_DST`; see Cross-Trigger Matrix.

### Architecture

Figure 100 is an example of a SiP with a heterogeneous cross triggering scheme utilizing both wire-OR and point-to-point cross triggering interfaces:

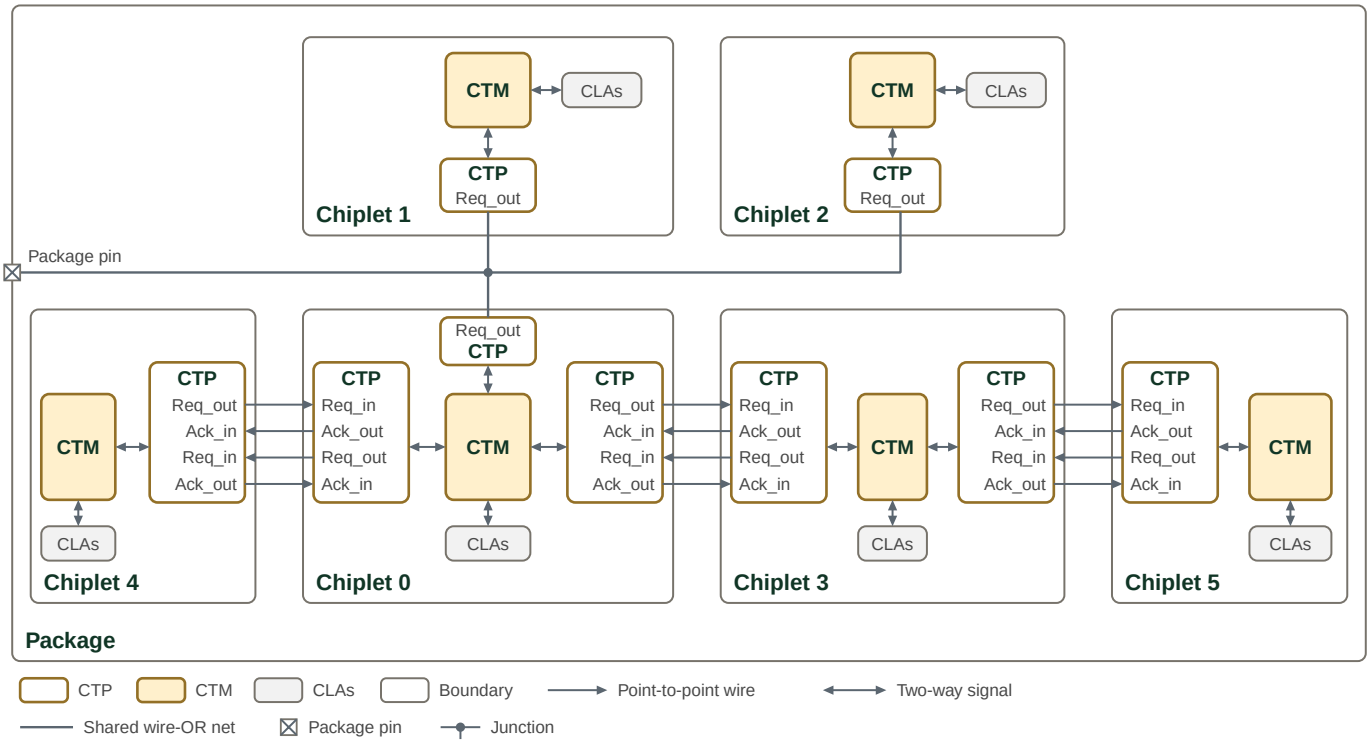


*Figure 100. SiP cross-trigger topology*

The package carries a single cross trigger across six chiplets. Chiplets 0, 1 and 2 each have a CTP in wire-OR mode on one shared wire, and point-to-point CTP pairs link Chiplet 0 with Chiplets 3 and 4, and Chiplet 3 with Chiplet 5. Chiplet 0 is the bridge between the wire-OR group and the point-to-point chiplets. Each CTP lists its pads on the side that faces its partner, in the order that keeps every point-to-point wire straight. The wire-OR net also reaches a package pin, which makes the trigger visible outside the package and could accept an external trigger; a package pin is impractical for point-to-point signalling.

The Programmer's Guide lists the CTM routing that each chiplet programs for this topology.

No route leads from a CTP back to itself, but a CTP in wire-OR mode also receives the falling edge that it drives onto the shared wire. A trigger that Chiplet 0 forwards to its wire-OR CTP therefore comes back from that CTP one wire-OR round trip later, and Chiplet 0's wire-OR rule forwards it again. For a trigger from Chiplet 3, the echo reaches Chiplet 0's local CLAs and Chiplets 3 and 4 a second time, and Chiplet 3 passes it on to Chiplet 5. Triggers from Chiplet 4, Chiplet 5 and Chiplet 0's local CLAs echo in the same way; triggers that enter from Chiplets 1 and 2 do not. Any bridge between a wire-OR group and point-to-point links behaves like this, so a destination can see one trigger twice, the second copy one wire-OR round trip after the first.

#### Cross Trigger Port (CTP)

The CTN supports from 1 to 32 external CTPs, set by its `NUM_CTP` parameter, which defaults to 16. Each external CTP uses one GPIO pad in wire-OR mode and two or four in point-to-point mode. These ports support both point-to-point and wire-OR cross trigger signaling modes.

Each CTP connects to four separate I/O pads:

- **CT_Req_out** - The cross trigger request output signal. The CTP asserts this signal to transmit a cross trigger in both point-to-point and wire-OR modes. In point-to-point mode the request is held high until the receiver acknowledges the request. In wire-OR mode, this signal drives the output-enable port of the I/O pad for duration defined by a pulse stretching module, and the pad output is driven static-low. Also in wire-OR mode the I/O pad input connects to the CTP without inversion. The shared wire is an open-drain active-low signal, and the CTP receives a cross trigger on its falling (assertion) edge.
- **CT_Ack_in** - The cross trigger acknowledge input signal. Exclusive to point-to-point mode. The cross trigger receiver asserts this signal when it acknowledges the CTP's cross trigger request.
- **CT_Req_in** - The cross trigger request input signal. Exclusive to point-to-point mode. This is the external cross trigger request received by the CTP.
- **CT_Ack_out** - The cross trigger acknowledge output signal. Exclusive to point-to-point mode. The CTP asserts this signal after receiving a cross trigger request, and de-asserts this signal after the cross trigger request is released.

When a GPIO pad is configured for cross triggering, the I/O pad assumes the role of one of these four cross trigger signals, and the pad controls and I/O data signals are connected to the associated CTP.

The logical structure of a CTP module is shown in Figure 101 in the Cross-Trigger Port architecture.

Signal names, directions, and pad controls are listed in the CTP Signal Interface.

The register interface for each CTP module is described in the CTN Memory Map section below.

#### Wire-OR Mode

Wire-OR mode is a 1-wire cross trigger signaling scheme compatible with many existing cross triggering implementations. This scheme is very resource efficient, as it only requires a single pin per chiplet connected to the network. However, the performance of the network decreases as the number of connected chiplets increase. Also, the

duration of the cross trigger pulse must be pre-arranged across all connected chiplets.

Each chiplet has an open-drain I/O connected to a single shared wire. Any connected chiplet may pull this signal low to assert a cross trigger. Depending on the electrical requirements, one or more internal pad pull-ups or external keepers ensure the cross trigger request signal remains high when no cross trigger is asserted. To ensure no connected chiplet misses a cross trigger pulse, all chiplets transmitting a trigger pulse must stretch the pulse a mutually agreed upon length of time.

Note that in wire-OR mode the *CT_Req_out* signal is connected to an open-drain configured I/O pad. For I/O pads that do have explicit open-drain control, the request data signal should connect to the output-enable control pin of the I/O pad, and the output data signal of the I/O pin should be driven statically low.

The logical behavior of a CTP in wire-OR mode is shown in Figure 102 in the Cross-Trigger Port architecture, and any number of chiplets may have their *CT_Req_out* pads interconnected as shown in Figure 103 in the same architecture section.

**Point-to-Point Mode**

Point-to-point mode utilizes a four-phase handshaking scheme to synchronize a cross trigger quickly and reliably between two chiplets without pulse stretching. While this scheme is simple and fast, it requires a dedicated CTP for each chiplet-to-chiplet interconnection, and each CTP requires four I/O pads for bi-directional communication, or two pads for unidirectional communication. All output pads are configured for push-pull operation.

The logical behavior of a CTP in point-to-point mode is shown in Figure 104 in the Cross-Trigger Port architecture, and any two chiplets may have their four cross trigger pads cross-coupled as shown in Figure 105 in the same architecture section.

On the sender side, the receiver's *CT_Ack_out* is connected to the sender's *CT_Ack_in* pad. *CT_Req_out* asserts whenever *CT_Src* asserts. *CT_Req_out* will then remain asserted until *CT_Ack_in* asserts.

On the receiver side, the sender's *CT_Req_out* is connected to the receiver's *CT_Req_in* pad. On any positive edge of *CT_Req_in*, the receiver generates a pulse on *CT_Dst* and asserts *CT_Ack_out*. Deassertion proceeds in exactly the same sequence.

The timing of the request/acknowledge handshake is shown in Figure 106 in the Cross-Trigger Port architecture.

**Cross Trigger Matrix (CTM)**

The CTM is a crossbar that aggregates and statically routes cross trigger pulses between various sources and sinks. Typical connections to the CTM include Core Logic Analyzers (CLAs) and CTPs, including other miscellaneous devices. The CTM is compatible with CTPs operating in either of the two signaling modes, point-to-point or wire-OR. Typically, each device or logic module connected to the CTM will act as both a source and a sink for cross trigger pulses.

The source selection and destination routing performed by the CTM are shown in Figure 107 in the Cross-Trigger Matrix architecture.

The CTM is programmed at runtime via firmware or debug transactions. Proper configuration of the CTMs in each chiplet will minimize trigger latency and prevent loops in the SiP cross trigger network. Configuration is accomplished via CSRs. The CTM register interface is described in the CTN Memory Map section below, and the Programmer's Guide gives the order in which to program the CTPs and the CTM.

### Address Map

The CTN provides a unified CSR address space accessed via a single AXI-Lite subordinate port. An internal AXI-Lite crossbar (`axi_lite_xbar`) routes transactions to the CTM and external CTPs based on address decoding. The crossbar registers all five channels at the subordinate port, so no combinational path runs from the port's inputs to its outputs, and each channel adds one cycle of latency. Each channel's register holds up to two beats and keeps its READY high while it holds fewer than two. The crossbar serves one CSR access at a time, so the port can accept a second request on a channel while the crossbar is still serving the first.

Table 2505 lists the CTM and external-CTP address ranges.

*Table 2505. CTN address allocation*

| Component | Address Range | Size | Description |
|---|---|---|---|
| CTM | 0x0000 - 0x01FF | 512 bytes | Cross Trigger Matrix configuration registers (one 8-byte `CT_SRC[k].CONFIG_0` slot per CT_Src port, up to 64 ports) |
| CTP[0] | 0x0200 - 0x020F | 16 bytes | External CTP 0 configuration |
| CTP[1] | 0x0210 - 0x021F | 16 bytes | External CTP 1 configuration |
| … | … | … | … |
| CTP[15] | 0x02F0 - 0x02FF | 16 bytes | External CTP 15 configuration |
| Internal CTPs | No CSR address | N/A | Configured via build-time parameters only |

The table shows the default 16 external CTPs. External CTP *n* occupies the 16 bytes from `0x200 + 0x10*n`.

The crossbar is configured with `MaxMstTrans=1` and `MaxSlvTrans=1`, serializing all CSR accesses. This simplification reduces arbitration logic at the cost of concurrent access capability.

### Internal vs External CTP Architecture

The CTN instantiates `NUM_CTP` external CTPs, 16 by default, using the full `cross_trigger_port` module with CSR interface, and `NUM_INT_CT` internal CTPs, 10 by default, using only the `cross_trigger_port_core` module without registers.

Table 2506 compares the modules, configuration, and connections used for external and internal CTPs.

*Table 2506. External and internal CTP comparison*

| Aspect | External CTP | Internal CTP |
|---|---|---|
| Module | `cross_trigger_port` (full) | `cross_trigger_port_core` (CSR-less) |
| CSR Interface | 16-byte AXI-Lite address space | None (build-time parameters) |
| GPIO Interface | 4-pad interface to die I/O | Logical signals to internal CLAs |
| Configuration | Runtime via CSRs (mode, invert, stretch_mult, reset) | Build-time via `INT_CT_MODE` parameter |
| Index Range | ctm_ct_src/dst[0:15] | ctm_ct_src/dst[16:25] |
| Signal Mapping | Standard GPIO pad controls | Repurposed GPIO signals (see Table 2507 and Table 2508) |

### Internal CTP Signal Remapping

Internal CTPs reuse the cross_trigger_port_core GPIO interface but connect to logical signals rather than physical pads. The `INT_CT_MODE` parameter bits are inverted before use: `ModeWireOr = ~INT_CT_MODE[i]`. This means `INT_CT_MODE[i]=0` selects Wire-OR mode, while `INT_CT_MODE[i]=1` selects Point-to-Point mode.

The internal wire-OR request from a CLA is active-high, while the CTP core receives a wire-OR cross trigger on the

falling edge of an idle-high wire. The network therefore inverts `ctm_dst_req_i[i]` before `ct_req_out_din`, so an internal CTP receives a cross trigger on the rising edge of the CLA request.

**Wire-OR Mode (INT_CT_MODE[i]=0, internal ModeWireOr=1):**

Table 2507 maps the CTP core signals to internal connections in wire-OR mode.

*Table 2507. Internal CTP wire-OR connections*

| CTP Core Signal | Internal Connection |
|---|---|
| `ct_req_out_dout_en` | → `ctm_src_req_o[i]` (stretched pulse to CLA) |
| `ct_req_out_din` | ← `~ctm_dst_req_i[i]` (stretched pulse from CLA, inverted) |
| All other signals | Unused (tied to constants) |

**Point-to-Point Mode (INT_CT_MODE[i]=1, internal ModeWireOr=0):**

Table 2508 maps the CTP core signals to internal connections in point-to-point mode.

*Table 2508. Internal CTP point-to-point connections*

| CTP Core Signal | Internal Connection |
|---|---|
| `ct_req_out_dout` | → `ctm_src_req_o[i]` (level-based request to CLA) |
| `ct_req_in_din` | ← `ctm_dst_req_i[i]` (level-based request from CLA) |
| `ct_ack_in_din` | ← `ctm_src_ack_i[i]` (acknowledge from CLA) |
| `ct_ack_out_dout` | → `ctm_dst_ack_o[i]` (acknowledge to CLA) |

This remapping allows internal CTPs to operate in either mode without physical GPIO pads, enabling on-chip cross-triggering between CLAs and the external cross-trigger network.

### Signal Flow

#### External Cross Trigger Flow

1. **Outgoing Trigger**: CTM ct_src → CTP ct_src_i → GPIO pads
2. **Incoming Trigger**: GPIO pads → CTP ct_dst_o → CTM ct_dst_i

#### Internal Cross Trigger Flow

1. **To Internal Sink**: CTM ct_src → Internal CTP → ctm_src_req_o
2. **From Internal Source**: ctm_dst_req_i → Internal CTP → CTM ct_dst

### Clock Stop Control

The `ctn_clock_stop_ctrl` module aggregates clock stop requests from multiple sources using a two-stage OR-tree architecture. The module separates CLA-only status from the combined functional halt signal to support independent JTAG visibility and chiplet clock gating control. The CTN sizes its request input with its own `NUM_CLK_STOP_REQ` parameter, which defaults to 9.

The parameters and ports of `ctn_clock_stop_ctrl` are listed in the HTML edition of this manual, under RTL Modules Reference.

The aggregation logic operates as follows:

1. **CLA OR-Tree**: All CLA request inputs are OR-reduced combinationally: `cla_clk_stop_req = |clk_stop_req_i`
2. **Combined Request**: JTAG and CLA requests are OR-combined: `clk_stop_req_combined = jtag_clock_stop_i | cla_clk_stop_req`
3. **Synchronization and Output Registration**: The combined request passes through a two-flop synchronizer into `clk_i` and then an output register, so `stop_clks_o` follows a change in either request after three `clk_i` cycles.

The synchronizer resolves the asynchronous JTAG and CLA requests into `clk_i`, and the output register keeps glitches out of the chiplet's functional clock gating logic. The combinatorial `cla_clock_stop_o` output provides immediate visibility of CLA activity to the JTAG interface, allowing debuggers to distinguish between JTAG-initiated and CLA-initiated clock stops.

#### Port Indexing

Table 2509 maps CTM port indices to external CTPs, internal CTPs, and clock-stop connections.

*Table 2509. CTN matrix port indices*

| Port Index | Description |
|---|---|
| 0 to N-1 | External CTPs (NUM_CTP) |
| N to N+M-1 | Internal CTPs (NUM_INT_CT) |

Where N = NUM_CTP and M = NUM_INT_CT. This indexing is used for CTM ct_dst_i and ct_src_o port assignment and CTM routing configuration registers.

#### Interface

#### Cross Trigger Port (CTP)

For the module diagram and signalling modes, see CTN Architecture.

Each CTP is built on `cross_trigger_port_core`. Its operating mode resets to wire-OR: the CSR MODE field of an external CTP resets to 0, which selects wire-OR, and 1 selects peer-to-peer. `INT_CT_MODE` sets the mode of each internal CTP.

The parameters and ports of `cross_trigger_port_core` are listed in the HTML edition of this manual, under RTL Modules Reference.

#### Cross Trigger Matrix (CTM)

The CTM has the following data ports and logical structure:

- **CT_Src[N-1:0]** - Cross trigger pulse output signals from the CTM to CLAs, CTPs, or other cross trigger sinking devices, where N is the number of these cross trigger sinks.
- **CT_Dst[M-1:0]** - Cross trigger pulse input signals to the CTM from CLAs, CTPs, or other cross trigger generating devices, where M is the number of these cross trigger sources.

#### Cross Trigger Network Module Interface

The parameters and ports of `cross_trigger_network` are listed in the HTML edition of this manual, under RTL Modules Reference.

### Dependencies

The CTN depends on the following components:

- `hw/ip/cross_trigger/cross_trigger_port` - Cross Trigger Port IP
- `hw/ip/cross_trigger/cross_trigger_matrix` - Cross Trigger Matrix IP
- `vendor/pulp-platform/axi` - AXI infrastructure (`axi_lite_xbar`)

### Programming

Cross-trigger configuration is static: every external CTP is configured once per boot before the CTM routes that use it are opened. The Programmer's Guide gives the cross-trigger programming sequence with reconfiguration and handshake deadlock recovery.

### Memory Map

### Memory Map

The Cross Trigger Network combines the Cross Trigger Matrix (CTM) and Cross Trigger Port (CTP) registers into a unified address space. Its generated map keeps the CTP instance count and stride with the allocated apertures:

*Table 2510. Cross Trigger Network Address Map*

| Base Address | End Address | Size | Decoded Extent | Unit | Instances | Stride | Description | Hole in Extent (R / W) | Past Extent (R / W) |
|---|---|---|---|---|---|---|---|---|---|
| 0x00000000 | 0x000001FF | 512 B | 208 B | Cross Trigger Matrix Address Map | 1 instance | - | Configuration registers for the cross trigger matrix | OKAY, 0x0 / OKAY | DECERR, 0xBADCAB1E / DECERR |
| 0x00000200 | 0x000002FF | 256 B | 252 B | Cross Trigger Port Address Map | 16 instances | 16 B | Configuration and status registers for the cross trigger port | OKAY, 0x0 / OKAY | OKAY, 0x0 / OKAY |

The CTN adds no CTN-specific registers; it decodes the AXI-Lite CSR interface onto the CTM and CTP blocks it instantiates. The composite SystemRDL top is the common source for these addresses and for the generated RTL constants.

The last two columns state how the network answers a 32-bit access that no register backs, as seen at its AXI-Lite CSR port, and Decode Response Codes defines them and their codes. An address above `0x2FF` matches no block and answers `DECERR`, with read data `0xBADCAB1E`.

### Detailed Register Map

### Cross Trigger Matrix Registers

### Cross Trigger Matrix Address Map

Configuration registers for the cross trigger matrix

| NOTE | **Register Arrays:** |
|---|---|

- **CT_SRC[26].CONFIG_0**: 26 registers, base 0x0, stride 0x8

*Table 2511. Cross Trigger Matrix Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xC8 | CT_SRC[26].CONFIG_0 | RW | Cross Trigger Source Configuration - CT_Dst select mask |

### CT_SRC[26].CONFIG_0

*Table 2512. cross_trigger_matrix.CT_SRC[26].CONFIG_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:26 | `Reserved` | - | - | Reserved |
| 25:0 | `CT_DST_SELECT` | RW | 0x0 | Selects the CT_Dst port(s) to forward on the CT_Src output at this array index. Each bit corresponds to one CT_Dst port, from CT_Dst[0] at bit 0 upwards. Multiple ports may be selected as a trigger source. All selected source ports are then OR'd together to produce the CT_Src output. Set this register to all-zero to disable that output. |

### Cross Trigger Port Registers

Each CTP window repeats the following register layout.

### Cross Trigger Port Address Map

Configuration and status registers for the cross trigger port

*Table 2513. Cross Trigger Port Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 | CONFIG | RW | Cross Trigger Port Configuration |
| 0x4 | STATUS | R | Cross Trigger Port Status. Writes complete with an OKAY response and leave the register unchanged. |
| 0x8 | STRETCH_MULT | RW | Pulse Stretch Multiplier |

### CONFIG

*Table 2514. cross_trigger_port.CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `RESET` | RW | 0x0 | While set, holds the outgoing point-to-point handshake idle with CT_Req_out deasserted and discards core-side cross trigger pulses for transmission. The incoming handshake keeps running. Set and then clear it to recover a handshake deadlock. |
| 1 | `INVERT` | RW | 0x0 | Inverts the sense of the incoming and outgoing GPIO signals. 0 - No inversion. Wire-OR mode uses active-low signaling with active or passive pull-ups. Point-to-point mode uses active-high signaling. 1 - All inputs and outputs of I/Os are inverted. Wire-OR mode uses active-high signaling with active or passive pull-downs. Point-to-point mode uses active-low signaling. Write it in the same access as MODE. |

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 0 | `MODE` | RW | 0x0 | Selects the operating mode of the CTP. 0 - Wire-OR, 1 - Point-to-Point. Program it once, in the same write as INVERT, from the package's cross trigger topology, before any cross trigger matrix route selects or targets this port. While wire-OR is selected the point-to-point handshake is held idle with its request deasserted; after point-to-point is selected, the handshake ignores the CT_Req_in and CT_Ack_in pads until their synchronized levels are valid. |

#### STATUS

*Table 2515. cross_trigger_port.STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `ACK_OUT` | R | 0x0 | Readout of the current CT_Ack_out level with CONFIG.INVERT undone, so 1 means the acknowledge is asserted whatever the pad polarity. Reads 0 in wire-OR mode. |
| 6 | `REQ_IN` | R | 0x0 | Readout of the current synchronized CT_Req_in level with CONFIG.INVERT undone, so 1 means the request is asserted whatever the pad polarity. |
| 5 | `ACK_IN` | R | 0x0 | Readout of the current synchronized CT_Ack_in level with CONFIG.INVERT undone, so 1 means the acknowledge is asserted whatever the pad polarity. |
| 4 | `REQ_OUT` | R | 0x0 | Readout of the current CT_Req_out level with CONFIG.INVERT undone, so 1 means the request is asserted whatever the pad polarity. Reads 0 in wire-OR mode. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `BUSY` | R | 0x0 | Indicates whether a pulse assertion or handshake is currently in progress. 0 - Pulse or handshake is not in progress, 1 - Pulse or handshake is currently in progress |

#### STRETCH_MULT

*Table 2516. cross_trigger_port.STRETCH_MULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `STRETCH_MULT` | RW | 0x0 | The number of clock cycles a core-side cross trigger pulse is stretched on the GPIO pin when in wire-OR mode. Whenever a core-side pulse is received, the generated GPIO pulse has a width of (STRETCH_MULT+1) clock cycles. |

### Composite Network Registers

This map gives the CTM and CTP register offsets within the generated CTN composition. The individual block maps above give offsets within each block.

### Cross Trigger Network Address Map

Cross trigger matrix and externally visible cross trigger ports

**NOTE**

**Register Arrays:**

- **CT_SRC[26].CONFIG_0**: 26 registers, base 0x0, stride 0x8
- **ctp[16].CONFIG**: 16 registers, base 0x200, stride 0x10

- **ctp[16].STATUS**: 16 registers, base 0x204, stride 0x10
- **ctp[16].STRETCH_MULT**: 16 registers, base 0x208, stride 0x10

*Table 2517. Cross Trigger Network Address Map register list*

| Address | Name | Access | Description |
|---|---|---|---|
| 0x0 - 0xC8 | CT_SRC[26].CONFIG_0 | RW | Cross Trigger Source Configuration - CT_Dst select mask |
| 0x200 - 0x2F0 | ctp[16].CONFIG | RW | Cross Trigger Port Configuration |
| 0x204 - 0x2F4 | ctp[16].STATUS | R | Cross Trigger Port Status. Writes complete with an OKAY response and leave the register unchanged. |
| 0x208 - 0x2F8 | ctp[16].STRETCH_MULT | RW | Pulse Stretch Multiplier |

### CT_SRC[26].CONFIG_0

*Table 2518. cross_trigger_network.CT_SRC[26].CONFIG_0 fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:26 | `Reserved` | - | - | Reserved |
| 25:0 | `CT_DST_SELECT` | RW | 0x0 | Selects the CT_Dst port(s) to forward on the CT_Src output at this array index. Each bit corresponds to one CT_Dst port, from CT_Dst[0] at bit 0 upwards. Multiple ports may be selected as a trigger source. All selected source ports are then OR'd together to produce the CT_Src output. Set this register to all-zero to disable that output. |

### ctp[16].CONFIG

*Table 2519. cross_trigger_network.ctp[16].CONFIG fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | `Reserved` | - | - | Reserved |
| 2 | `RESET` | RW | 0x0 | While set, holds the outgoing point-to-point handshake idle with CT_Req_out deasserted and discards core-side cross trigger pulses for transmission. The incoming handshake keeps running. Set and then clear it to recover a handshake deadlock. |
| 1 | `INVERT` | RW | 0x0 | Inverts the sense of the incoming and outgoing GPIO signals. 0 - No inversion. Wire-OR mode uses active-low signaling with active or passive pull-ups. Point-to-point mode uses active-high signaling. 1 - All inputs and outputs of I/Os are inverted. Wire-OR mode uses active-high signaling with active or passive pull-downs. Point-to-point mode uses active-low signaling. Write it in the same access as MODE. |
| 0 | `MODE` | RW | 0x0 | Selects the operating mode of the CTP. 0 - Wire-OR, 1 - Point-to-Point. Program it once, in the same write as INVERT, from the package's cross trigger topology, before any cross trigger matrix route selects or targets this port. While wire-OR is selected the point-to-point handshake is held idle with its request deasserted; after point-to-point is selected, the handshake ignores the CT_Req_in and CT_Ack_in pads until their synchronized levels are valid. |

### ctp[16].STATUS

*Table 2520. cross_trigger_network.ctp[16].STATUS fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | `Reserved` | - | - | Reserved |
| 7 | `ACK_OUT` | R | 0x0 | Readout of the current CT_Ack_out level with CONFIG.INVERT undone, so 1 means the acknowledge is asserted whatever the pad polarity. Reads 0 in wire-OR mode. |
| 6 | `REQ_IN` | R | 0x0 | Readout of the current synchronized CT_Req_in level with CONFIG.INVERT undone, so 1 means the request is asserted whatever the pad polarity. |
| 5 | `ACK_IN` | R | 0x0 | Readout of the current synchronized CT_Ack_in level with CONFIG.INVERT undone, so 1 means the acknowledge is asserted whatever the pad polarity. |
| 4 | `REQ_OUT` | R | 0x0 | Readout of the current CT_Req_out level with CONFIG.INVERT undone, so 1 means the request is asserted whatever the pad polarity. Reads 0 in wire-OR mode. |
| 3:1 | `Reserved` | - | - | Reserved |
| 0 | `BUSY` | R | 0x0 | Indicates whether a pulse assertion or handshake is currently in progress. 0 - Pulse or handshake is not in progress, 1 - Pulse or handshake is currently in progress |

#### ctp[16].STRETCH_MULT

*Table 2521. cross_trigger_network.ctp[16].STRETCH_MULT fields*

| Bits | Field | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | `Reserved` | - | - | Reserved |
| 15:0 | `STRETCH_MULT` | RW | 0x0 | The number of clock cycles a core-side cross trigger pulse is stretched on the GPIO pin when in wire-OR mode. Whenever a core-side pulse is received, the generated GPIO pulse has a width of (STRETCH_MULT+1) clock cycles. |

### Cross-Trigger Port

The Cross Trigger Port (CTP) is the pad-facing signalling module for cross-trigger communication per the Open Chiplet Cross Triggering (OCCT) specification. Each CTP connects the chiplet core logic to four I/O pads and handles the timing and handshaking required for both wire-OR and point-to-point signalling schemes.

#### Overview

Each CTP connects to four I/O pads:

- **CT_Req_out** — cross-trigger request output.
- **CT_Ack_in** — acknowledge input (point-to-point only).
- **CT_Req_in** — cross-trigger request input (point-to-point only).
- **CT_Ack_out** — acknowledge output (point-to-point only).

Two inter-chiplet signalling schemes are supported:

- **Wire-OR** — supports a wide range of existing chiplet implementations and is resource efficient. Uses pulse stretching to ensure reliable detection across chiplet boundary delays.
- **Point-to-point** — faster, more reliable, and simpler to configure. Uses four-phase handshaking with CDC synchronisers at the die boundary.

### Configuration

The CTP has no build-time parameters. All operational settings are programmed at runtime through the AXI-Lite CSR interface:

Table 2522 summarises the runtime controls exposed through the CTP CSR interface.

*Table 2522. CTP configuration registers*

| Register | Description |
|---|---|
| `MODE` | Select wire-OR (0) or point-to-point (1) signalling. |
| `STRETCH_MULT` | 16-bit pulse stretch multiplier for wire-OR mode. |
| `INVERT` | Signal polarity inversion for GPIO adaptation. |
| `RESET` | Handshake state machine reset. |

See the CTN Memory Map for the full CTP register layout and field descriptions. The CTN decodes the AXI-Lite CSR window and presents each CTP's registers at a 16-byte stride starting at offset 0x0200.

`MODE` and `INVERT` are static configuration, set once per boot in a single write before any CTM route selects or targets the port. The Programmer's Guide gives the cross-trigger programming sequence with reconfiguration and deadlock recovery.

### Architecture

The Cross Trigger Port architecture separates the register interface from core logic to enable reuse. The full `cross_trigger_port` module includes CSRs for runtime configuration, while the `cross_trigger_port_core` module provides the core functionality without registers, allowing instantiation for internal cross-trigger connections without AXI-Lite overhead.

Figure 101 shows the Cross Trigger Port logic and its pad interfaces.

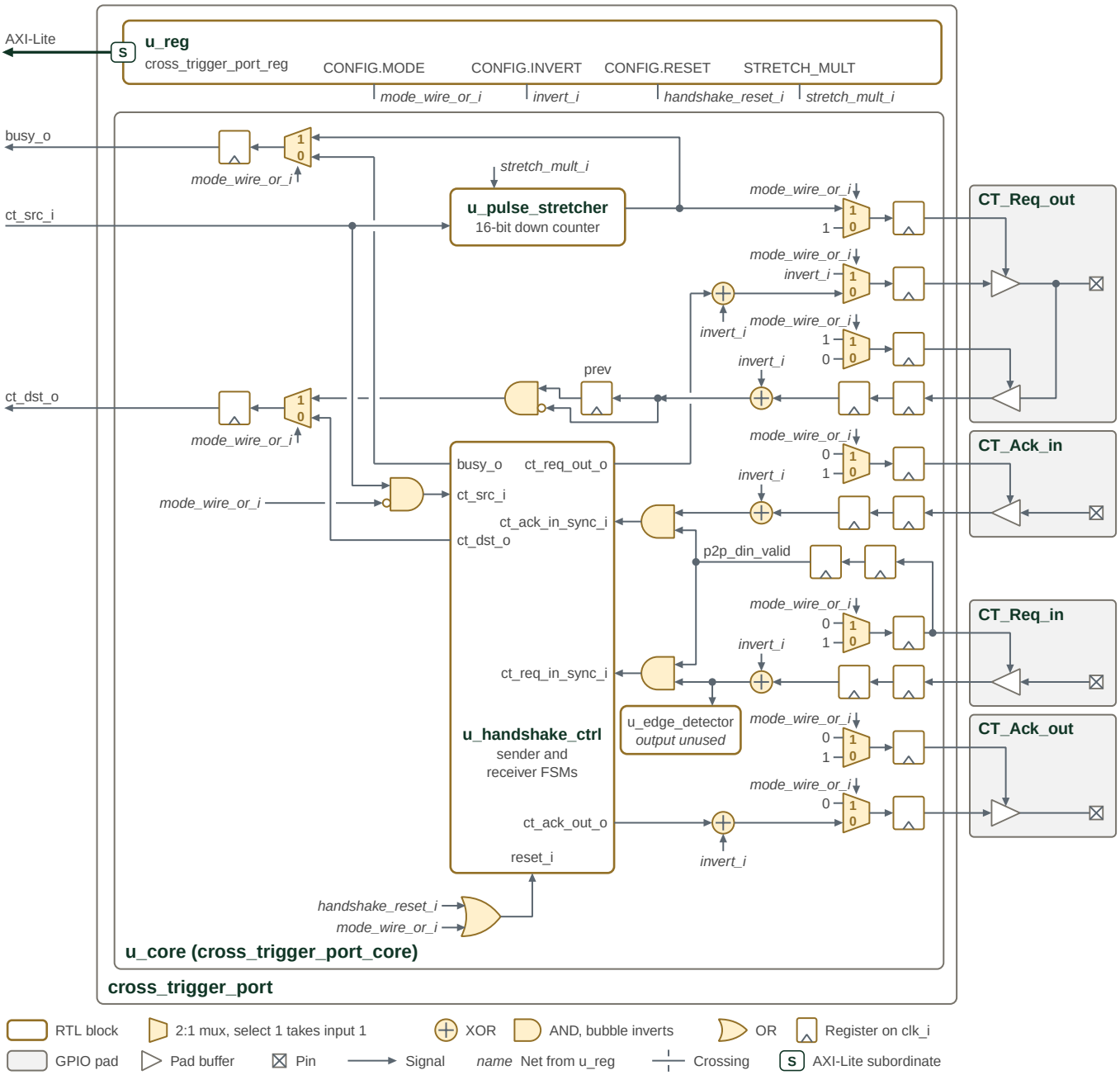


*Figure 101. CTP module structure*

The figure is read from the core side on the left to the four GPIO pads on the right. Every 2:1 mux is selected by `mode_wire_or_i`, and the numbers inside it mark which input each select value takes. Input 1, drawn upper, passes in wire-OR mode, when `mode_wire_or_i` is high because `CONFIG.MODE` is 0; input 0, drawn lower, passes in point-to-point mode. Italic names are nets driven by `u_reg` from the named CSR fields. Every pad control and both core-side outputs, `ct_dst_o` and `busy_o`, leave the port through a register on `clk_i`. Each pad input passes through two synchroniser registers and an XOR with `invert_i` and, on the handshake inputs, an AND with `p2p_din_valid` before any logic uses it. The `STATUS` read-back from the core to `u_reg` is not drawn.

### Submodule Hierarchy

The CTP core instantiates four specialized submodules for different aspects of cross-trigger operation:

Table 2523 identifies the submodules that implement synchronization and cross-trigger operation.

*Table 2523. CTP submodules*

| Module | Purpose |
|---|---|
| `ctp_synchronizer` | Implements 2-FF synchronizers for asynchronous GPIO inputs (ct_req_out_din, ct_req_in_din, ct_ack_in_din) from remote chiplets |
| `ctp_pulse_stretcher` | Extends source pulse duration for Wire-OR mode transmission, with configurable stretch multiplier (0-65535 cycles) |
| `ctp_edge_detector` | Detects positive edges on the synchronized ct_req_in signal; its output is not connected, because the receiver state machine works on the ct_req_in level |
| `ctp_handshake_ctrl` | Implements sender and receiver state machines for Point-to-Point mode four-phase handshaking protocol |

### Wire-OR Mode

Wire-OR mode is a 1-wire cross trigger signaling scheme compatible with many existing cross triggering implementations. This scheme is very resource efficient, as it only requires a single pin per chiplet connected to the network. However, the performance of the network decreases as the number of connected chiplets increase. Also, the duration of the cross trigger pulse must be pre-arranged across all connected chiplets.

### Pulse Stretcher Operation

The pulse stretcher extends single-cycle core-side pulses to multi-cycle GPIO assertions lasting `stretch_mult + 1` clock cycles. The module implements a 16-bit countdown counter that loads the stretch multiplier value when a new pulse arrives and decrements to zero. If a new pulse arrives before the counter reaches zero, the counter restarts from the stretch multiplier value, ensuring that rapid successive pulses merge into an extended assertion.

The output `stretched_pulse` remains asserted as long as the counter is non-zero. This stretched pulse drives the output enable control for the open-drain CT_Req_out GPIO pad, effectively pulling the shared wire low for the configured duration.

Each chiplet has an open-drain I/O connected to a single shared wire. Any connected chiplet may pull this signal low to assert a cross trigger. Depending on the electrical requirements, one or more internal pad pull-ups or external keepers ensure the cross trigger request signal remains high when no cross trigger is asserted. To ensure no connected chiplet misses a cross trigger pulse, all chiplets transmitting a trigger pulse must stretch the pulse a mutually agreed upon length of time.

Note that in wire-OR mode the *CT_Req_out* signal is connected to an open-drain configured I/O pad. For I/O pads that do have explicit open-drain control, the request data signal should connect to the output-enable control pin of the I/O pad, and the output data signal of the I/O pin should be driven statically low.

Figure 102 shows the open-drain request path and pulse-stretching logic used in wire-OR mode. It is Figure 101 with every mux resolved to its wire-OR input. The CT_Req_out pad drives the constant `invert_i`, so its output enable, the stretched pulse, decides when the pad pulls the wire. The CTP receives a cross trigger on the falling edge of the synchronised wire: the register marked *prev* holds the previous level and the AND gate fires when the previous level is high and the current one low. The CTP disables the buffers of the other three pads in this mode, so they are not drawn.

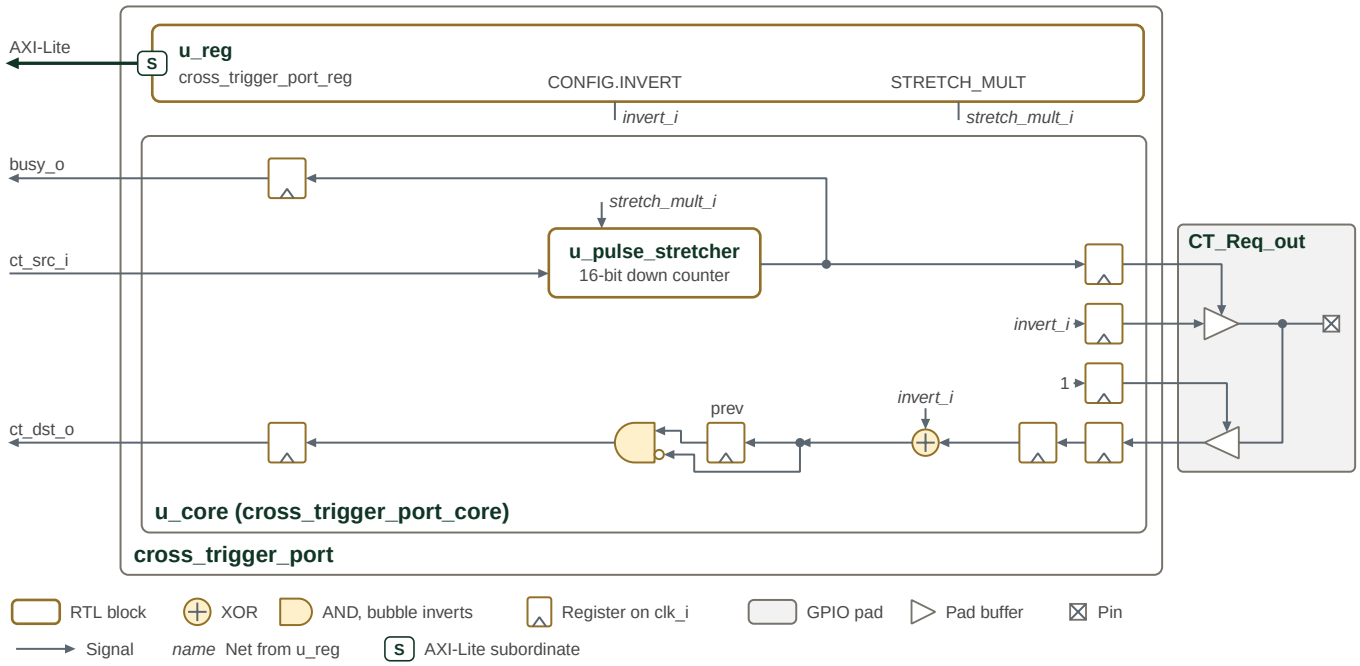


*Figure 102. CTP wire-OR behavior*

Any number of chiplets may have their *CT_Req_out* pads interconnected as shown in Figure 103. In each chiplet the CLAs reach the CTM through the internal CTPs of the Cross Trigger Network, and the CTM reaches the external CTP through its *CT_Src* and *CT_Dst* signals. The shared wire also reaches a package pin.

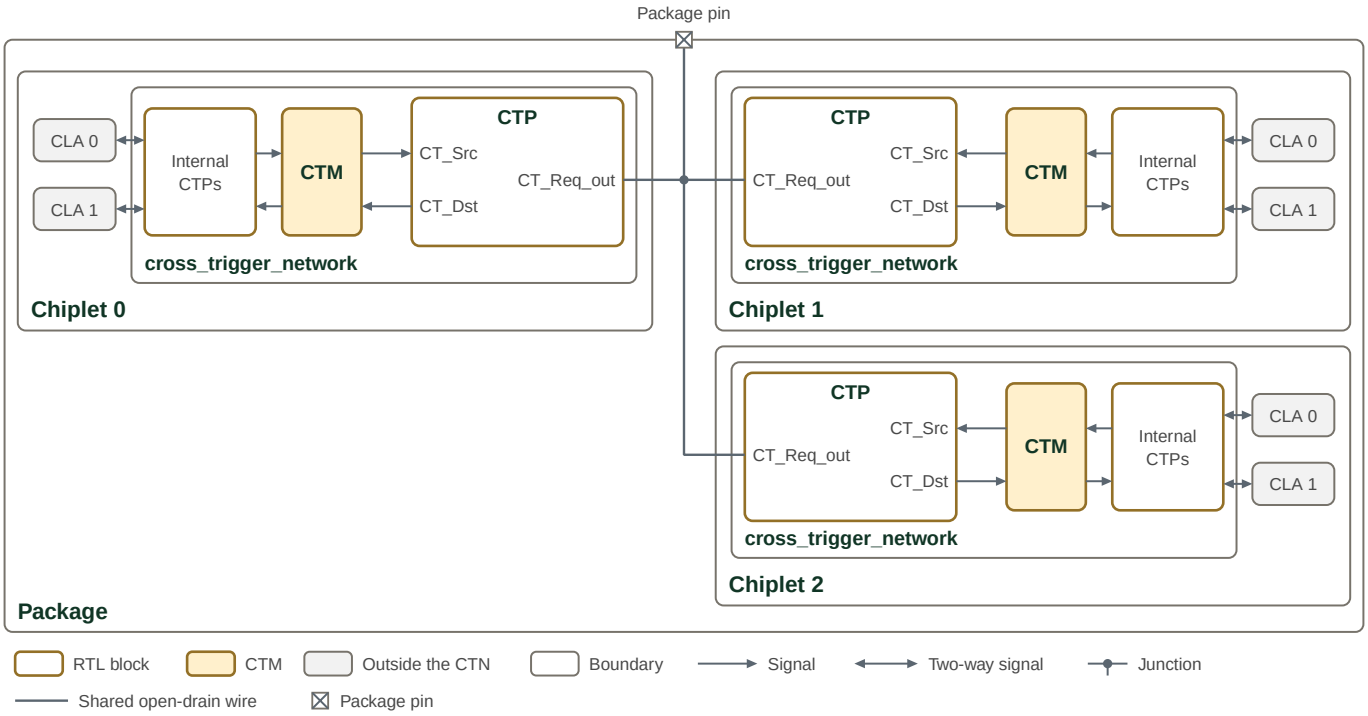


*Figure 103. CTP wire-OR chiplet connections*

### Point-to-Point Mode

Point-to-point mode utilizes a four-phase handshaking scheme to synchronize a cross trigger quickly and reliably between two chiplets without pulse stretching. While this scheme is simple and fast, it requires a dedicated CTP for each

chiplet-to-chiplet interconnection, and each CTP requires four I/O pads for bi-directional communication, or two pads for unidirectional communication. All output pads are configured for push-pull operation.

Figure 104 shows the request and acknowledge paths used for point-to-point handshaking. It is Figure 101 with every mux resolved to its point-to-point input. The output enables of CT_Req_out and CT_Ack_out and the input enables of CT_Ack_in and CT_Req_in are registered 1s. The input enable of the CT_Req_out pad is 0 in this mode, so only its output buffer is drawn, and the pulse stretcher and falling-edge detector are unused.

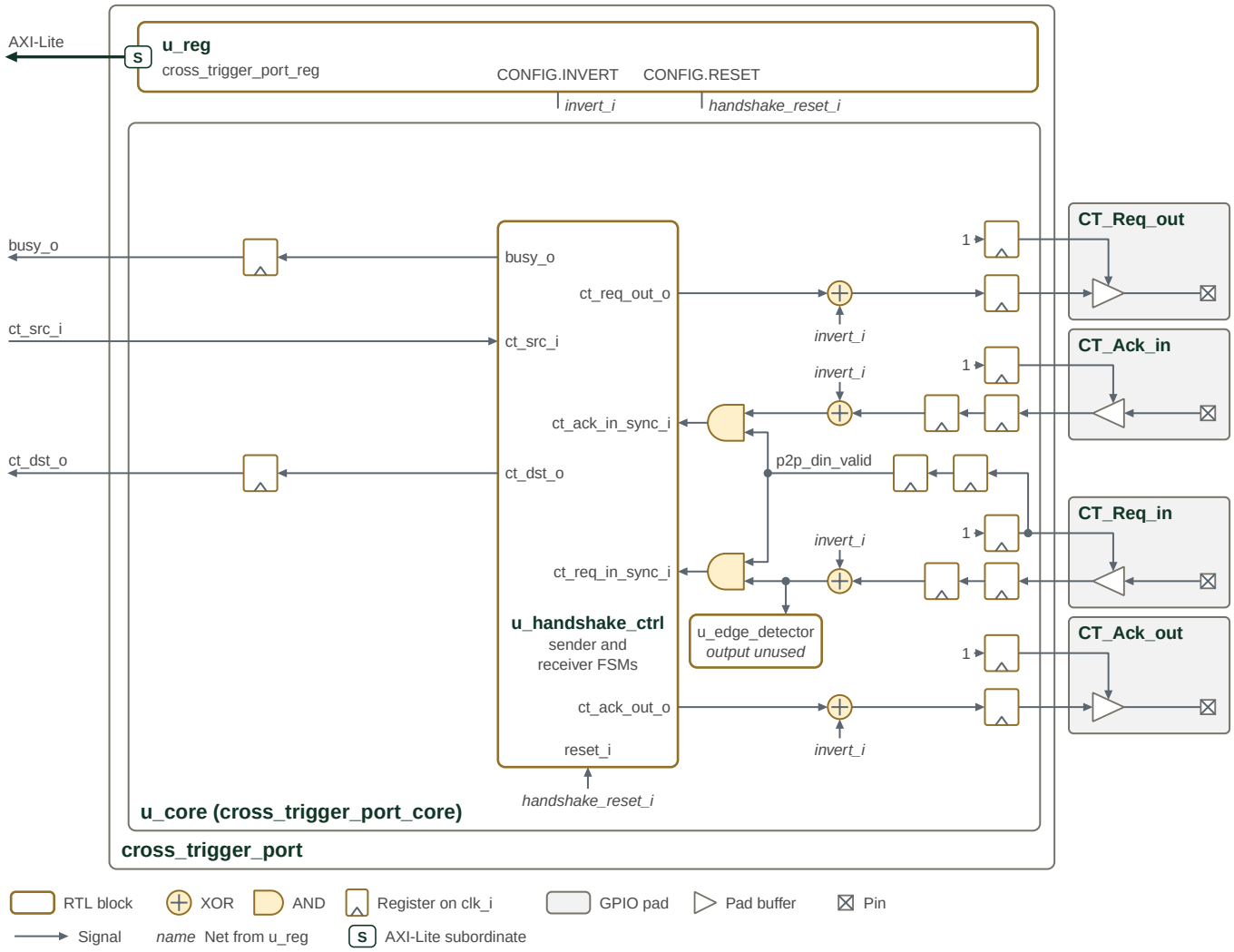


*Figure 104. CTP point-to-point behavior*

Any two chiplets may have their four cross trigger pads cross-coupled as shown in Figure 105. Each CTP lists its pads on the side that faces its partner, in the order that keeps every wire straight, so each request output meets a request input and each acknowledge output an acknowledge input.

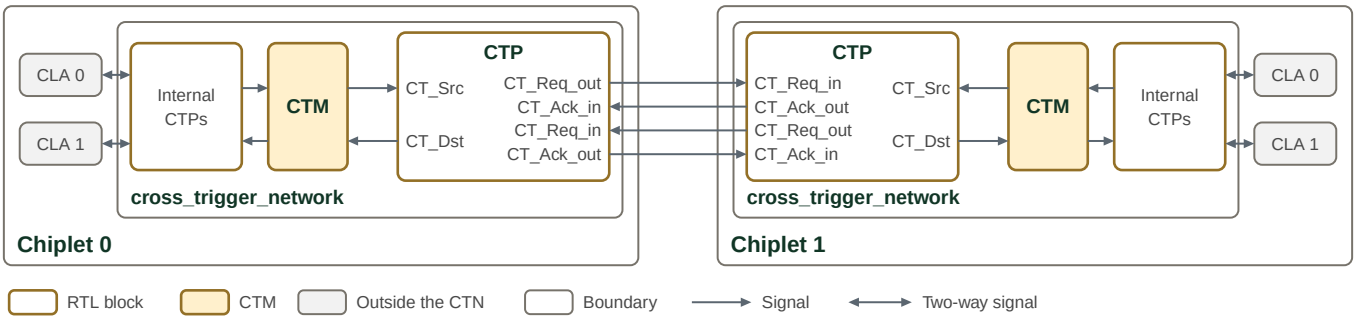


*Figure 105. CTP point-to-point chiplet connections*

On the sender side, the receiver's *CT_Ack_out* is connected to the sender's *CT_Ack_in* pad. *CT_Req_out* asserts whenever *CT_Src* asserts. *CT_Req_out* will then remain asserted until *CT_Ack_in* asserts.

On the receiver side, the sender's *CT_Req_out* is connected to the receiver's *CT_Req_in* pad. On any positive edge of *CT_Req_in*, the receiver generates a pulse on *CT_Dst* and asserts *CT_Ack_out*. Deassertion proceeds in exactly the same sequence.

The timing of the request/acknowledge handshake is shown in Figure 106. The timing is illustrative, and the delays are not to scale:

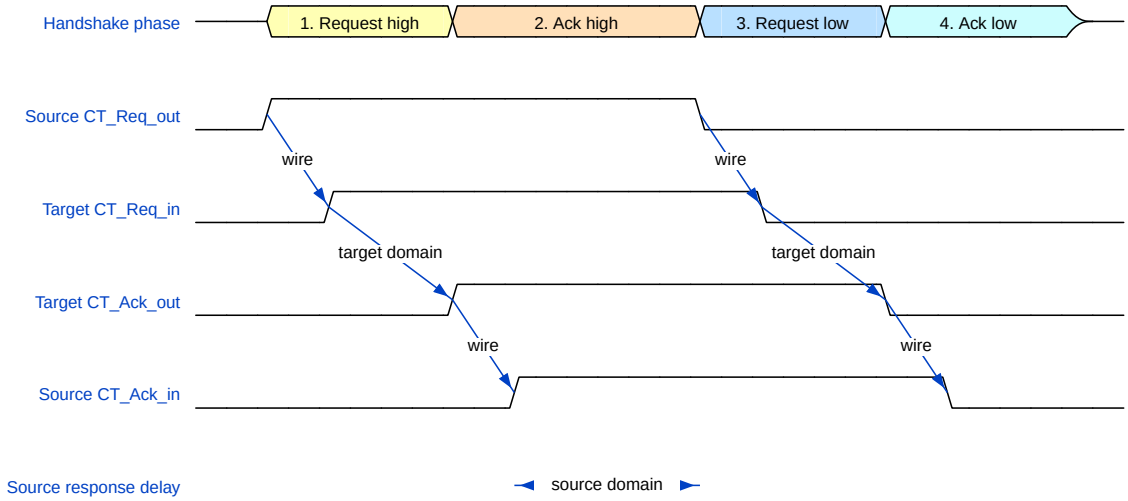


*Figure 106. CTP request-acknowledge timing*

### Synchronization and CDC Handling

All GPIO input signals are asynchronous with respect to the CTP system clock and require synchronization to prevent metastability. The `ctp_synchronizer` module implements three independent 2-FF synchronizer chains using the `prim_flop_2sync` primitive:

Table 2524 identifies the incoming signals synchronized in each operating mode.

*Table 2524. CTP signal synchronization*

| Signal | Used In Mode | Synchronization Purpose |
|---|---|---|
| `ct_req_out_din` | Wire-OR | Synchronizes open-drain wire state from remote chiplets before edge detection |
| `ct_req_in_din` | Point-to-Point | Synchronizes request signal from remote sender before receiver FSM |

| Signal | Used In Mode | Synchronization Purpose |
|---|---|---|
| `ct_ack_in_din` | Point-to-Point | Synchronizes acknowledge signal from remote receiver before sender FSM |

After synchronization, an optional inversion is applied if the `INVERT` configuration bit is set. This allows adaptation to GPIO pads with different polarity requirements. Edge detection and state machine processing occur only on synchronized (and optionally inverted) signals, ensuring all CTP core logic operates on glitch-free inputs.

#### Handshake State Machines

Point-to-point mode implements two independent state machines within the `ctp_handshake_ctrl` module, enabling full-duplex cross-triggering. The sender state machine controls outgoing cross-trigger requests, while the receiver state machine handles incoming requests. Both state machines operate synchronously in the system clock domain on synchronized GPIO inputs.

The state machines run only in point-to-point mode. While wire-OR mode is selected, `cross_trigger_port_core` holds the sender in SENDER_IDLE with `ct_req_out=0` and blocks `ct_src_i` from reaching it. The `CT_Req_in` and `CT_Ack_in` pad inputs are disabled in wire-OR mode, and after point-to-point mode is selected their synchronizers hold the disabled pad level for two cycles. The core presents both synchronized inputs to the state machines as deasserted until the delayed input enable shows the synchronizers carry pad data. Neither a trigger raised in wire-OR mode nor the level of a disabled pad reaches the point-to-point pads when the mode changes.

**Sender State Machine:**

Table 2525 lists the sender handshake states, transition conditions, and output actions.

*Table 2525. CTP sender state transitions*

| Current State | Condition | Next State | Output Actions |
|---|---|---|---|
| SENDER_IDLE | `ct_src_i` asserted | SENDER_REQ_ASSERTED | Assert `ct_req_out=1` |
| SENDER_REQ_ASSERTED | `ct_ack_in_sync` asserted | SENDER_WAIT_ACK_DEASSERT | De-assert `ct_req_out=0` |
| SENDER_WAIT_ACK_DEASSERT | `ct_ack_in_sync` de-asserted | SENDER_IDLE | None (remain idle) |
| Any state | `reset_i` asserted | SENDER_IDLE | Force `ct_req_out=0` |
| Any state | Wire-OR mode selected | SENDER_IDLE | Force `ct_req_out=0` |

**Receiver State Machine:**

Table 2526 lists the receiver handshake states, transition conditions, and output actions.

*Table 2526. CTP receiver state transitions*

| Current State | Condition | Next State | Output Actions |
|---|---|---|---|
| RECEIVER_IDLE | `ct_req_in_sync` asserted | RECEIVER_ACK_ASSERTED | Assert `ct_ack_out=1`, generate `ct_dst` pulse |
| RECEIVER_ACK_ASSERTED | `ct_req_in_sync` de-asserted | RECEIVER_WAIT_REQ_DEASSERT | De-assert `ct_ack_out=0`, clear `ct_dst` |
| RECEIVER_WAIT_REQ_DEASSERT | `ct_req_in_sync` remains low | RECEIVER_IDLE | None |
| RECEIVER_WAIT_REQ_DEASSERT | `ct_req_in_sync` goes high | Remain in state | Continue waiting |

The `ct_dst` output pulse is generated for a single clock cycle when the receiver state machine transitions from IDLE to ACK_ASSERTED. The busy signal indicates an active cross-trigger transfer: it reflects either state machine being outside its IDLE state. It is registered once in `ctp_handshake_ctrl` and again in `cross_trigger_port_core`, so `busy_o` and `STATUS.BUSY` rise two `clk_i` cycles after a state machine leaves IDLE and fall two `clk_i` cycles after both state machines return to

IDLE. Logic that samples `busy_o` within two cycles of a trigger on `ct_src_i`, or of the end of a handshake, sees the previous value.

### Interface

#### Signal Interface

Packed control and bus types remain bundles at the `cross_trigger_port` module boundary; their members are not separate scalar ports.

The parameters and ports of `cross_trigger_port` are listed in the HTML edition of this manual, under RTL Modules Reference.

`cross_trigger_port` includes the CSR interface. The register-free `cross_trigger_port_core` instead takes `mode_wire_or_i`, `invert_i`, `handshake_reset_i`, and `stretch_mult_i` as configuration inputs. At that core interface, `mode_wire_or_i = 1` selects wire-OR and `0` selects point-to-point; the CSR MODE field is inverted before reaching it.

### Dependencies

The CTP depends on the following components:

- `hw/common/ocah_prim` - Primitive cells (synchronizers, pulse stretcher)

### Cross-Trigger Matrix

The Cross Trigger Matrix (CTM) is an AND-OR crossbar that aggregates and statically routes cross-trigger pulses between all sources (CTPs, CLAs, internal devices) and sinks under AXI-Lite CSR control. The default DTP configuration uses 26 input ports and 26 output ports.

### Overview

The CTM is compatible with both wire-OR and point-to-point CTPs. Typical connections include Core Logic Analyzers (CLAs) and Cross Trigger Ports (CTPs). Each device connected to the CTM acts as both a source and a sink.

The `NUM_CT_SRC` and `NUM_CT_DST` parameters in `regs/cross_trigger_matrix.rdl` define the matrix dimensions and default to 26 each. The RTL derives the following localparams from the generated register map:

Table 2527 lists the source and destination counts used by the CTM register map.

*Table 2527. CTM configuration constants*

| Constant | Value | Description |
|---|---|---|
| `NUM_CT_SRC` | 26 | Number of CT_Src output ports (sized by the `CT_SRC` array in the register map). |
| `NUM_CT_DST` | 26 | Number of CT_Dst input ports (sized by the `CT_DST_SELECT` field width in the register map). |

These values cannot be overridden at instantiation. Resizing the matrix requires regenerating the register map with PeakRDL addrmap overrides (`OCAH_REG_RDL_PARAMS_cross_trigger_matrix`) that match the elaborated Cross-Trigger Network total (`NUM_CTP + NUM_INT_CT`). The CTN fails elaboration if that total disagrees with the generated `NUM_CT_SRC` and `NUM_CT_DST`. The CTM takes the AXI-Lite request and response types as module parameters, which the CTN overrides to match its own crossbar.

The wrapper passes the register block only the address bits its register map spans, and the CTN ends the matrix's crossbar window at the end of that map. An access past the last CT_Src register completes with DECERR rather than

reaching a register. The 512-byte CTM aperture holds 64 source registers at the 8-byte stride, so a matrix of up to 64 ports needs only the regenerated CTM and CTN register maps.

Every CT_Src port occupies 8 bytes regardless of the port count. At or below 32 destinations, each register is a single 32-bit word with immediate write semantics. Above 32 destinations both words must be written before the mask takes effect, preventing half-programmed states.

The register map for the CTM and for each CTP is included once in the CTN Memory Map section. This page covers the crossbar architecture and signal interface only.

### Architecture

The Cross Trigger Matrix has N source ports and N destination ports, one of each per CTM port of the Cross Trigger Network, so N = `XTRIG_NUM_CTP` + `XTRIG_NUM_INT_CT`. The `dtp_pkg` defaults of 16 external CTPs and 10 internal CTPs give N = 26. `cross_trigger_matrix_pkg` takes `NUM_CT_SRC` and `NUM_CT_DST` from the register map, and the CTN asserts at elaboration that both equal `NUM_CTP` + `NUM_INT_CT`, so resizing the matrix means regenerating `regs/cross_trigger_matrix.rdl` to match. The matrix implements a combinatorial AND-OR crossbar with registered outputs to route cross-trigger pulses between CTPs and internal cross-trigger sources.

Figure 107 shows how the CTM selects and combines sources for each registered destination.

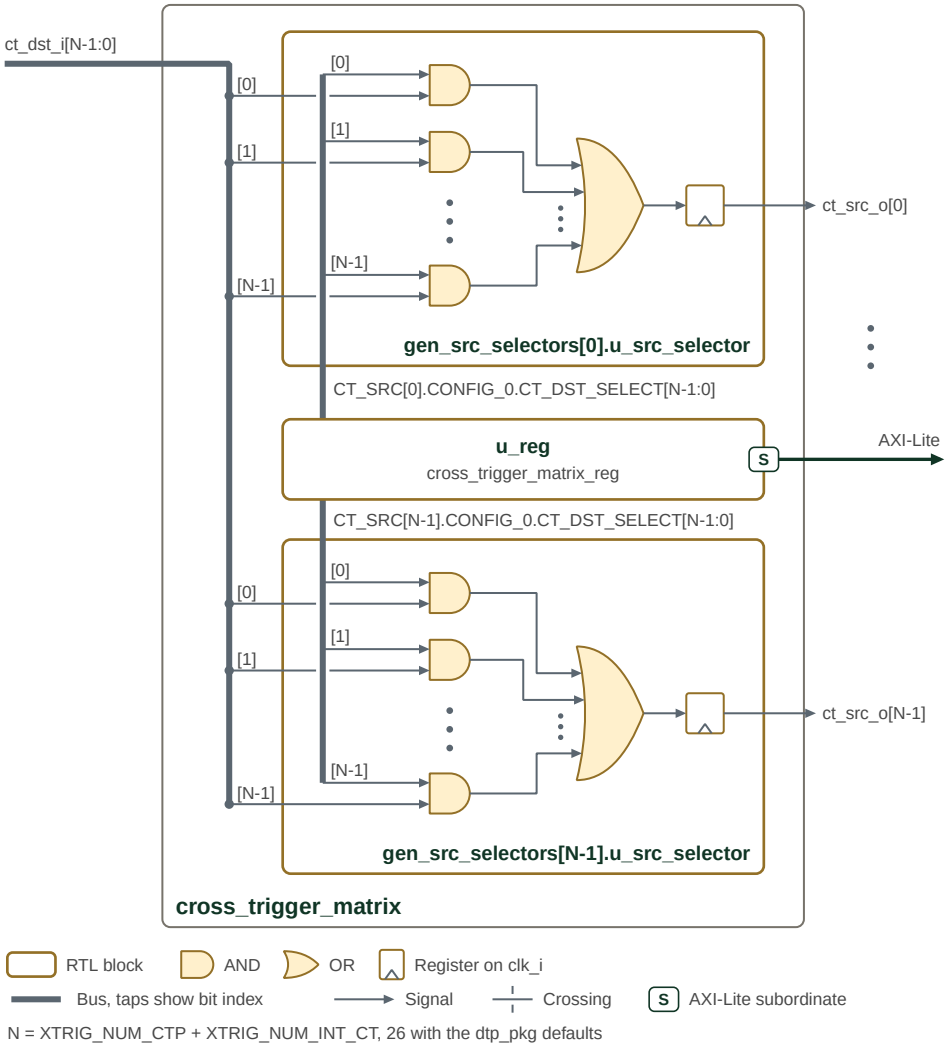


*Figure 107. CTM routing structure*

The figure draws two of the N selectors, for `ct_src_o[0]` and `ct_src_o[N-1]`, and in each the AND gates for inputs 0, 1 and N-1; the dots stand for the rest. The thick lines are buses whose taps carry the bit index: the `ct_dst_i` bus fans out to every selector, and each selector reads its own `CT_SRC[j].CONFIG_0.CT_DST_SELECT` field from `u_reg`, the matrix CSR block reached over AXI-Lite.

The CTM is programmed at runtime via firmware or debug transactions. Proper configuration of the CTMs in each chiplet will minimize trigger latency and prevent loops in the SiP cross trigger network.

### Routing Logic

The CTM routing implements a two-stage combinatorial path followed by output registration. Each of the N CT_Src output ports is driven by a dedicated `ctm_src_selector` module that performs the following operation:

Table 2528 describes the selection, combination, and register stages of destination routing.

*Table 2528. CTM routing stages*

| Stage | Operation |
|---|---|
| Selection (AND) | Each CT_Dst[i] input is AND-gated with its corresponding select bit: `selected_pulses[i] = ct_dst_i[i] & select_mask[i]` |
| Aggregation (OR) | All selected pulses are OR-reduced: `ct_src_comb = \|selected_pulses` |
| Registration | The combinatorial result is registered on `clk_i` rising edge: `ct_src_o ⇐ ct_src_comb` |

This architecture allows each CT_Src output to combine pulses from multiple CT_Dst inputs through OR-aggregation. Multiple CT_Src outputs can source from the same CT_Dst input, enabling broadcast distribution. The register stage prevents glitches on the output when configuration changes or when multiple input pulses arrive simultaneously.

### Port Indexing

Table 2529 maps matrix indices to external CTPs, internal CTPs, and clock-stop connections.

*Table 2529. CTM port indices*

| Port Index | Description |
|---|---|
| 0 to NUM_CTP-1 | External CTPs (NUM_CTP) |
| NUM_CTP to N-1 | Internal CTPs (NUM_INT_CT) |

NUM_CTP and NUM_INT_CT are the CTN parameters that `XTRIG_NUM_CTP` and `XTRIG_NUM_INT_CT` set. This indexing is used for CTM ct_dst_i and ct_src_o port assignment and CTM routing configuration registers.

### Interface

### Signal Interface

Packed control and bus types remain bundles at the `cross_trigger_matrix` module boundary; their members are not separate scalar ports.

The parameters and ports of `cross_trigger_matrix` are listed in the HTML edition of this manual, under RTL Modules Reference.

`NUM_CT_SRC` and `NUM_CT_DST` come from the generated register package. Routing masks are programmed through the AXI-Lite CSR port; there are no separate route-configuration or acknowledge ports.

#### Data Ports

The CTM has the following data ports:

- **CT_Src[N-1:0]** - Cross trigger pulse output signals from the CTM to CLAs, CTPs, or other cross trigger sinking devices, where N is the number of these cross trigger sinks.
- **CT_Dst[M-1:0]** - Cross trigger pulse input signals to the CTM from CLAs, CTPs, or other cross trigger generating devices, where M is the number of these cross trigger sources.

#### Dependencies

The CTM depends on the following components:

- `hw/common/ocah_prim` - Primitive cells

### 2.1.7.7. Configuration and Reference

This page provides the DTP top-level port table. Configuration parameters are defined on their owning IP pages; see the links in the section below. The generated CTM and CTP register maps appear in the CTN Memory Map section of the Cross-Trigger Network page.

#### Port Reference

Table 2530 lists all DTP top-level ports. The port table is shared with the Integrator Guide.

*Legend: Where no Default entry is shown in Description / Default, the original port reference specifies N/A — those ports have no integrator-supplied default; connect as directed by Description.*

Table 2530 lists the DTP boundary ports, their dimensions, and their connections.

*Table 2530. DTP Port Declaration*

| Interface Name | Type | Dir | Width | Description / Default | Connection |
|---|---|---|---|---|---|
| `clk_i` | `logic` | In | 1 | System clock for cross trigger logic and JTAG2AXI bridges. | System clock |
| `rst_n_i` | `logic` | In | 1 | System reset (active low). Resets cross trigger and AXI logic; does not affect the TAP controller. | System reset |
| `pwr_on_rst_ni` | `logic` | In | 1 | Power-on reset for JTAG state machine (active low, asynchronous). | Power-on reset |
| `dbg_disable_i` | `sep_lifecycle_ctrl_pkg::dbg_disable_t` | In | - | Per-path lifecycle controls for disabling debug and test interfaces. Active-high (1 = path disabled). DTP synchronizes each field into the JTAG TCK domain. | SEP lifecycle controller or adopter-equivalent policy source |
| `jtag_ptap_client_tap_ctrl_i` | `jtag_tap_ctrl_t` | In | - | Primary JTAG TAP control (TCK, TMS, TRST_N). | External JTAG pads |
| `jtag_ptap_client_tdi_i` | `logic` | In | 1 | Primary JTAG TDI. | External JTAG pads |
| `jtag_ptap_client_tdo_o` | `logic` | Out | 1 | Primary JTAG TDO. | External JTAG pads |
| `jtag_ptap_client_tdo_oen_o` | `logic` | Out | 1 | Primary JTAG TDO output enable. | External JTAG pads |
| `jtag_bsr_host_scan_ctrl_o` | `jtag_scan_ctrl_t` | Out | - | Boundary scan control. Tie scan_in to scan_out if BSR is disabled. | Boundary scan chain |
| `jtag_bsr_host_scan_in_i` | `logic` | In | 1 | Boundary scan data in. | Boundary scan chain |

| Interface Name | Type | Dir | Width | Description / Default | Connection |
|---|---|---|---|---|---|
| jtag_bsr_host_scan_out_o | logic | Out | 1 | Boundary scan data out. | Boundary scan chain |
| jtag_stap_io_host_tap_ctrl_o | jtag_tap_ctrl_t | Out | - | I/O STAP TAP control for chiplet-to-chiplet connectivity. If JTAG_STAP_IO_ENABLE=0, only .tck is driven; .tms is 0 and .trst_n is 0, so TRST is asserted and a TAP connected to the port is held in Test-Logic-Reset. | I/O STAP host |
| jtag_stap_io_host_tdi_i | logic | In | 1 | I/O STAP TDI. Default: Tie to '0 if unused. | I/O STAP host |
| jtag_stap_io_host_tdo_o | logic | Out | 1 | I/O STAP TDO. | I/O STAP host |
| jtag_stap_io_host_tdo_oen_o | logic | Out | 1 | I/O STAP TDO output enable. | I/O STAP host |
| jtag_stap_smc_host_tap_ctrl_o | jtag_tap_ctrl_t | Out | - | SMC debug STAP TAP control. If JTAG_SMC_DBG_ENABLE=0, only .tck is driven; .tms is 0 and .trst_n is 0, so TRST is asserted and a TAP connected to the port is held in Test-Logic-Reset. | SMC JTAG port |
| jtag_stap_smc_host_tdi_i | logic | In | 1 | SMC debug STAP TDI (from SMC TDO). Default: Tie to '0 if unused. | SMC JTAG port |
| jtag_stap_smc_host_tdo_o | logic | Out | 1 | SMC debug STAP TDO (to SMC TDI). | SMC JTAG port |
| jtag_stap_smc_host_tdo_oen_o | logic | Out | 1 | SMC debug STAP TDO output enable. | SMC JTAG port |
| jtag_stap_sep_host_tap_ctrl_o | jtag_tap_ctrl_t | Out | - | SEP debug STAP TAP control. If JTAG_SEP_DBG_ENABLE=0, only .tck is driven; .tms is 0 and .trst_n is 0, so TRST is asserted and a TAP connected to the port is held in Test-Logic-Reset. | SEP JTAG port |
| jtag_stap_sep_host_tdi_i | logic | In | 1 | SEP debug STAP TDI (from SEP TDO). Default: Tie to '0 if unused. | SEP JTAG port |
| jtag_stap_sep_host_tdo_o | logic | Out | 1 | SEP debug STAP TDO (to SEP TDI). | SEP JTAG port |
| jtag_stap_sep_host_tdo_oen_o | logic | Out | 1 | SEP debug STAP TDO output enable. | SEP JTAG port |
| jtag_stap_extra_host_tap_ctrl_o | jtag_tap_ctrl_t | Out | [JtagNumExtraStapPorts-1:0] | Extra STAP TAP control array. If JTAG_NUM_EXTRA_STAPS=0, the single padded element drives only .tck; its .tms is 0 and its .trst_n is 0, so TRST is asserted and a TAP connected to it is held in Test-Logic-Reset. | Extra STAP hosts |
| jtag_stap_extra_host_tdi_i | logic | In | [JtagNumExtraStapPorts-1:0] | Extra STAP TDI array. Default: Tie to '0 if unused. | Extra STAP hosts |
| jtag_stap_extra_host_tdo_o | logic | Out | [JtagNumExtraStapPorts-1:0] | Extra STAP TDO array. | Extra STAP hosts |
| jtag_stap_extra_host_tdo_oen_o | logic | Out | [JtagNumExtraStapPorts-1:0] | Extra STAP TDO output enable array. | Extra STAP hosts |
| jtag_stap_host_scan_ctrl_o | jtag_scan_ctrl_t | Out | - | Extended STAP scan control. Tie scan_in to scan_out if unused. | STAP scan chain |
| jtag_stap_host_scan_in_i | logic | In | 1 | Extended STAP scan data in. | STAP scan chain |
| jtag_stap_host_scan_out_o | logic | Out | 1 | Extended STAP scan data out. | STAP scan chain |
| jtag_dfd_host_scan_ctrl_o | jtag_scan_ctrl_t | Out | - | DFD iJTAG scan control. Tie scan_in to scan_out if unused. | DFD iJTAG chain |
| jtag_dfd_host_scan_in_i | logic | In | 1 | DFD iJTAG scan data in. | DFD iJTAG chain |
| jtag_dfd_host_scan_out_o | logic | Out | 1 | DFD iJTAG scan data out. | DFD iJTAG chain |

| Interface Name | Type | Dir | Width | Description / Default | Connection |
|---|---|---|---|---|---|
| `jtag_dft_secure_host_scan_ctrl_o` | `jtag_scan_ctrl_t` | Out | - | Secure DFT iJTAG scan control. Tie scan_in to scan_out if unused. | Secure DFT iJTAG chain |
| `jtag_dft_secure_host_scan_in_i` | `logic` | In | 1 | Secure DFT iJTAG scan data in. | Secure DFT iJTAG chain |
| `jtag_dft_secure_host_scan_out_o` | `logic` | Out | 1 | Secure DFT iJTAG scan data out. | Secure DFT iJTAG chain |
| `jtag_dft_host_scan_ctrl_o` | `jtag_scan_ctrl_t` | Out | - | Non-secure DFT iJTAG scan control. Tie scan_in to scan_out if unused. | Non-secure DFT iJTAG chain |
| `jtag_dft_host_scan_in_i` | `logic` | In | 1 | Non-secure DFT iJTAG scan data in. | Non-secure DFT iJTAG chain |
| `jtag_dft_host_scan_out_o` | `logic` | Out | 1 | Non-secure DFT iJTAG scan data out. | Non-secure DFT iJTAG chain |
| `axi_smc_dbg_req_o` | `smc_jtag_axi_req_t` | Out | - | SMC fabric debug AXI request. Active when `JTAG_SMC_DBG_ENABLE=1`. Default: tie to '0 if disabled. | SMC `jtag_axi_in_req_i` |
| `axi_smc_dbg_resp_i` | `smc_jtag_axi_resp_t` | In | - | SMC fabric debug AXI response. Default: tie to '0 if disabled. | SMC `jtag_axi_in_resp_o` |
| `axil_smc_otp_jtag_req_o` | `smc_otp_axil_req_t` | Out | - | SMC OTP debug AXI-Lite request. Default: tie to '0 if disabled. | SMC `axil_smc_otp_jtag_req_i` |
| `axil_smc_otp_jtag_resp_i` | `smc_otp_axil_resp_t` | In | - | SMC OTP debug AXI-Lite response. Default: tie to '0 if disabled. | SMC `axil_smc_otp_jtag_resp_o` |
| `axil_sep_otp_jtag_req_o` | `sep_otp_axil_req_t` | Out | - | SEP OTP debug AXI-Lite request. Default: tie to '0 if disabled. | SEP `axil_sep_otp_jtag_req_i` |
| `axil_sep_otp_jtag_resp_i` | `sep_otp_axil_resp_t` | In | - | SEP OTP debug AXI-Lite response. Default: tie to '0 if disabled. | SEP `axil_sep_otp_jtag_resp_o` |
| `stop_clks_o` | `logic` | Out | 1 | Clock stop output from JTAG DEBUG_CONTROL or CLA clock-stop requests via CTN. | PLL clock gates |
| `cla_clock_stop_en_o` | `logic` | Out | 1 | CLA clock stop enable from DEBUG_CONTROL register. | Clock control logic |
| `jtag_boot_stall_ovrd_o` | `logic` | Out | 1 | JTAG boot stall override enable. | Boot control |
| `jtag_boot_stall_o` | `logic` | Out | 1 | JTAG boot stall value when override is asserted. | Boot control |
| `jtag_ic_reset_smc_o` | `ic_reset_smc_t` | Out | `$bits(ic_reset_smc_t)` | SMC IC_RESET override slice. Struct contains `.ovrd` (override enables) and `.val` (active-low override values), each sized to match the corresponding SMC reset control struct. Tied to '0 when `JTAG_IC_RESET_SMC_ENABLE=0`. | SMC `jtag_smc_reset_ctrl_i` |
| `jtag_ic_reset_sep_o` | `ic_reset_sep_t` | Out | `$bits(ic_reset_sep_t)` | SEP IC_RESET override slice. Struct contains `.ovrd` (override enables) and `.val` (active-low override values), each sized to match the corresponding SEP reset control struct. Tied to '0 when `JTAG_IC_RESET_SEP_ENABLE=0`. | SEP `jtag_sep_reset_ctrl_i` |
| `jtag_ic_reset_ext_o` | `ic_reset_ext_t` | Out | `$bits(ic_reset_ext_t)` | External IC_RESET override slice. Struct contains `.ovrd` (override enables) and `.val` (active-low override values), each sized per integrator-defined type. Tied to '0 when `JTAG_IC_RESET_EXT_ENABLE=0`. | Integrator-defined reset overrides for logic outside the SMU |
| `jtag_ptap_state_o` | `tap_state_e` | Out | - | Current PTAP TAP state. | Debug infrastructure |
| `jtag_ptap_inst_decoded_o` | `jtag_instruction_decoded_e` | Out | - | Current decoded PTAP instruction. | Debug infrastructure |

| Interface Name | Type | Dir | Width | Description / Default | Connection |
|---|---|---|---|---|---|
| axil_xtrig_req_i | xtrig_axil_req_t | In | - | Cross trigger CSR AXI-Lite request from SMC. | SMC axil_dtp_csr_req_o |
| axil_xtrig_resp_o | xtrig_axil_resp_t | Out | - | Cross trigger CSR AXI-Lite response to SMC. | SMC axil_dtp_csr_resp_i |
| xtrig_ctm_src_req_o | logic | Out | [XTRIG_NUM_INT_CT-1:0] | Cross trigger source requests from CTM to destination sinks. | Cross trigger sinks |
| xtrig_ctm_src_ack_i | logic | In | [XTRIG_NUM_INT_CT-1:0] | Cross trigger source acknowledges. Unused in pulse-sync mode (mode bit = 0). Default: tie to '0 in pulse-sync mode. | Cross trigger sinks |
| xtrig_ctm_dst_req_i | logic | In | [XTRIG_NUM_INT_CT-1:0] | Cross trigger destination requests from source generators. Default: tie to '0 if unused. | Cross trigger sources |
| xtrig_ctm_dst_ack_o | logic | Out | [XTRIG_NUM_INT_CT-1:0] | Cross trigger destination acknowledges. Unused in pulse-sync mode (mode bit = 0). | Cross trigger sources |
| xtrig_clk_stop_req_i | logic | In | [XTRIG_NUM_CLK_STOP_REQ-1:0] | Clock stop requests from CLAs or other devices in the chiplet. Default: Tie to '0 if unused. | Clock stop requesters |
| xtrig_ctp_req_out_dout_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-out data output (to pad ring). | GPIO pad ring |
| xtrig_ctp_req_out_dout_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-out output enable. | GPIO pad ring |
| xtrig_ctp_req_out_din_i | logic | In | [XTRIG_NUM_CTP-1:0] | CTP request-out loopback input. Default: tie to '0 if unused. | GPIO pad ring |
| xtrig_ctp_req_out_din_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-out input enable. | GPIO pad ring |
| xtrig_ctp_req_in_dout_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-in data output. | GPIO pad ring |
| xtrig_ctp_req_in_dout_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-in output enable. | GPIO pad ring |
| xtrig_ctp_req_in_din_i | logic | In | [XTRIG_NUM_CTP-1:0] | CTP request-in data input. Default: tie to '0 if unused. | GPIO pad ring |
| xtrig_ctp_req_in_din_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP request-in input enable. | GPIO pad ring |
| xtrig_ctp_ack_in_dout_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-in data output. | GPIO pad ring |
| xtrig_ctp_ack_in_dout_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-in output enable. | GPIO pad ring |
| xtrig_ctp_ack_in_din_i | logic | In | [XTRIG_NUM_CTP-1:0] | CTP ack-in data input. Default: tie to '0 if unused. | GPIO pad ring |
| xtrig_ctp_ack_in_din_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-in input enable. | GPIO pad ring |
| xtrig_ctp_ack_out_dout_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-out data output. | GPIO pad ring |
| xtrig_ctp_ack_out_dout_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-out output enable. | GPIO pad ring |
| xtrig_ctp_ack_out_din_i | logic | In | [XTRIG_NUM_CTP-1:0] | CTP ack-out data input. Default: tie to '0 if unused. | GPIO pad ring |
| xtrig_ctp_ack_out_din_en_o | logic | Out | [XTRIG_NUM_CTP-1:0] | CTP ack-out input enable. | GPIO pad ring |
| test_en_i | logic | In | 1 | DFT test-mode enable (active high) for the JTAG2AXI bridges and the cross trigger CSR crossbar. Default: 0 in functional mode. | DFT controller |

| Interface Name | Type | Dir | Width | Description / Default | Connection |
|---|---|---|---|---|---|
| `scan_rst_ni` | `logic` | In | 1 | DFT scan reset (active low). Unused: DTP has no reset synchronizer, so `rst_n_i` must arrive scan-controlled. | DFT controller, or tie high |

**Configuration Parameters**

DTP instantiation parameters are defined on their owning IP pages:

- **JTAG Interface Unit** parameters (STAP count, I/O/SMC/SEP/extra STAP enables) — see the JTAG Interface Unit page.
- **PTAP** parameters (IC_RESET struct types, IDCODE fields, JTAG2AXI pipeline depths, feature-enable enables) — see the JTAG PTAP page.
- **Cross Trigger Network** parameters (CTP count, internal CT count, clock-stop request count, per-CTP operating mode) — see the Cross-Trigger Network page.

**Address Map**

CTN Register Map

# 2.2. System Management Network (SMN)

The System Management Network (SMN) is the system fabric outside the SMU. It carries transactions between the SMU and resources elsewhere in the chiplet. SEP, SMC and DTP belong to the SMU; the SMN is a separate platform component.

## 2.2.1. SMU Interface

The SMU exposes two AXI interfaces toward the SMN:

- `smu_axi_in_req_i` / `smu_axi_in_resp_o`: external requests entering the SMU and their responses.
- `smu_axi_out_req_o` / `smu_axi_out_resp_i`: requests leaving the SMU and responses from the external fabric.

The internal SMU crossbar connects SEP, SMC and these external interfaces. It routes incoming requests to the programmed SEP or SMC global aperture. SEP and SMC requests that match neither aperture leave through the external output. External requests cannot loop back to that output; an unmatched incoming address produces a decode error.

## 2.2.2. Addressing and Integration

SEP and SMC global base addresses and region sizes are software-programmed. The surrounding system must use the same address assignments when accessing those subsystems. Subsystem remapping and filtering are described in their Interconnect chapters, and the Programmer's Guide gives the aperture programming sequence.

This chapter describes the SMU boundary. Network topology, routing beyond that boundary and connections to other chiplets depend on the enclosing system.

See SMU AXI Crossbar, OCAH Memory Map, SEP Interconnect and SMC Interconnect.

# 2.3. AXI-over-UCIe Bridge (AoU)

The AXI-over-UCIe bridge transports AXI transactions between chiplets through the UCIe interface. These pages reproduce the AOU_CORE Microarchitecture Specification, originally authored by BOS Semiconductors, and follow the AXI over UCIe protocol specification.

## 2.3.1. Overview and Features

Apache License Release
AOU_CORE Microarchitecture specification
Version 1.00, Feb.27 2026
BOS Semiconductors

### 2.3.1.1. Introduction

This microarchitecture specification includes detailed functional description of AOU_CORE.

The contents of this document are given as follows. Section 1 is for overview. Section 2 is for features, implementation specific features, and assumptions. In Section 3, the architecture and operation of AoU IP are given. Section 4 covers interrupt sources. Section 5 covers error handling. Section 6 includes the integration guidelines such as clocks, resets, and I/O descriptions. Section 7 includes the software operation guide including the register map and activation flows. Section 8 includes performance, power, and area information.

**Overview**

The AOU_CORE is a bridge between AXI interface and the UCIe FDI interface, defined in AoU standard v0.7 spec. To achieve low latency, AOU_CORE directly handles signals related to the UCIe FDI interface data flow control and processes data in 32-byte, 64-byte, or 128-byte chunks (selected at integration time via the FDI_CONFIG parameter; see the Integrator guide) in a cut-through manner, without converting them to 256-byte flit data. It receives AXI messages from its own AXI subordinate interface, packs them into chunks of the configured FDI width, and transmits these chunks to the remote AoU device via UCIe. The remote AoU receives the chunks from the UCIe FDI interface, unpacks them into AXI messages, and delivers the data through its AXI manager interface. The one-way AoU latency, measured from the AXI input of the local device to the AXI output of the remote device over a back-to-back connection in a single clock domain, is 7 cycles for read requests, 8 cycles for read data, 9 cycles for write requests, 9 cycles for write data, and 6 cycles for write responses. For applications using UCIe, including those with some CDC domains, the round-trip time (RTT) latency from an AXI read request to the corresponding AXI read response through this AoU IP over UCIe is approximately 45 cycles, excluding the memory latency of the remote device.

To achieve high efficiency, all TX requests and data messages are packed in a compact format, ensuring that no AoU payload slots are wasted. The current payload is even updated when the UCIe stalls the current chunk and space remains available. As a result, the throughput matches the values defined in the AoU specification. In each cycle, the RX can process the maximum number of messages defined in the AoU standard: 4 read requests, 4 write requests, 2 read data, 2 write data, and 12 write responses, respectively.

It supports flow control for all exceptional cases defined in the UCIe standard. First, it handles the chunk valid/cancel signal, including the alternative implementation method described in the UCIe specification. Second, it handles the stall request signal received from the D2D Adapter while maintaining the Flit-aligned boundary. Interoperability is also supported when the remote device uses a different AXI data width.

AOU_CORE supports a configurable number of Resource Plane(RPs), with both the number of RPs and each RP's AXI data width fully parameterizable. A dedicated field in the SFR allows software to configure the target RP, enabling flexible RP

mapping. For each RP, the AW, W and AR channel support three arbitration modes: Round Robin, AXI QoS scheme, and Port QoS scheme. Additionally, a starvation-prevention mechanism is implemented to ensure fair arbitration across all RP channels.

The operating frequency of AOU_CORE targets 1 GHz and meets timing requirements without relying on multi-cycle paths. The TX buffer is configured with a minimal number of FIFO entries, while the RX Data buffer default depth is FDI-config aware: default 140 entries when FDI_CONFIG selects a 1024b (128B) FDI interface (FDI_CFG_SP_128B or FDI_CFG_TP_64B_128B), and default 88 entries otherwise. The entry size takes into account the round-trip latency for credit consumption and return, and matches the per-RP RP*_RX_W_FIFO_DEPTH / RP*_RX_R_FIFO_DEPTH parameter defaults documented in the Integrator guide.

### 2.3.1.2. Features

Figure 108 shows the AXI-to-UCIe bridge functions and their relationships.

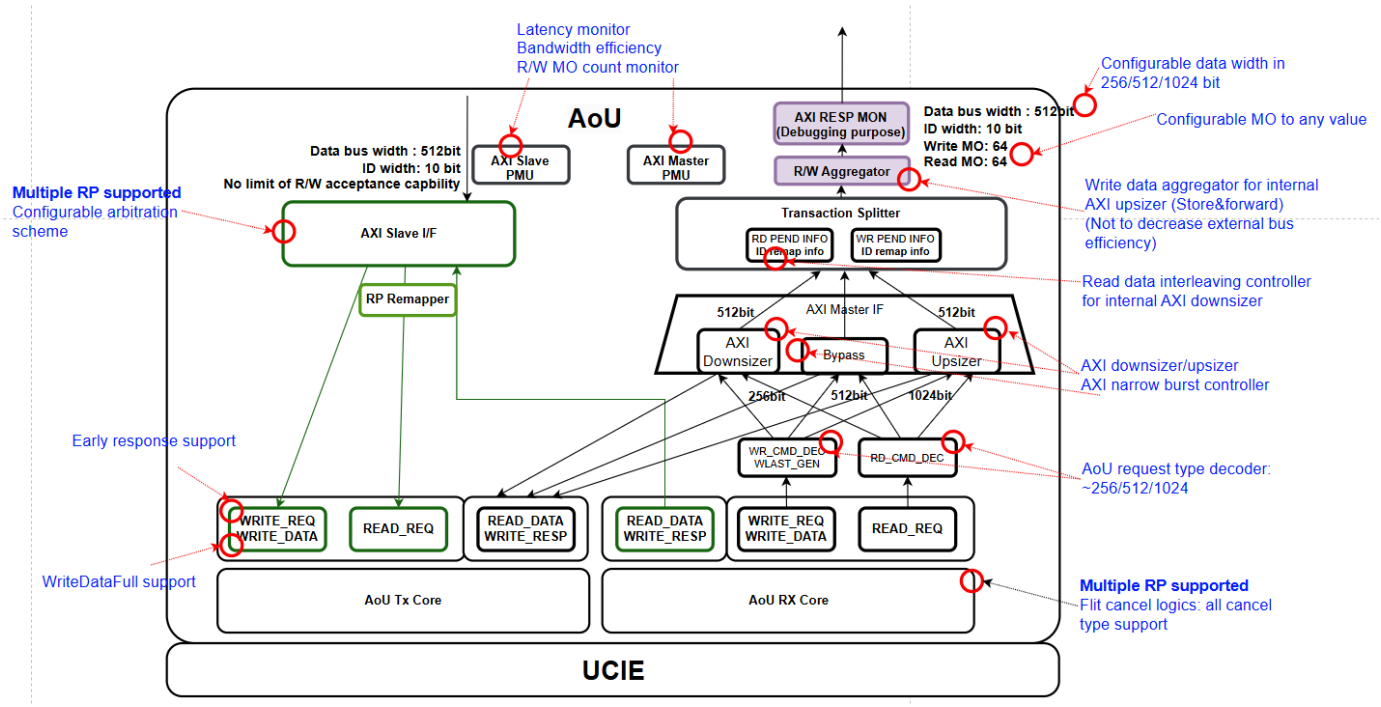


*Figure 108. Block Diagram of AOU_CORE and function description*

**Basic features**

- Single APB subordinate interface
  - Configuration/control/status register access
- Multiple AXI subordinate & manager interface
  - Configurable RP port & RP remapping supported
  - Supports variable data widths and burst length requests. Configurable FIFO Depth
- UCIe 256B latency-optimized flit(Format 6) support only
- Message packing and unpacking are handled in a unit of 32, 64, or 128 bytes, corresponding to the configured FDI data width (32B@1 GHz, 64B@1 GHz, or 128B@1 GHz).
- Selectable Early response by setting SFR
- Selectable 32B / 64B / 128B single-PHY and 32B+64B / 64B+128B two-PHY FDI configurations, selected via the

FDI_CONFIG parameter (see the Integrator guide). Two-PHY variants additionally require the +define+TWO_PHY build flag so the second-PHY ports physically exist.

- Configurable number of Resource Plane and each RP's AXI data width
- RP Remapping
- QoS scheme between RP with AW, W, AR channel
- Interoperability with different data width remote die
- Handling of flit cancel and stall request as specified in the UCIe specification
- Selectable AXI Aggregator added for BUS efficiency
- According to the AoU specification, except for the AXI data width, the parameters AXI_ADDR_WD, AXI_ID_WD, and AXI_LEN_WD are fixed. Porting to match the width of the connected ports must be handled externally to AOU_CORE_TOP.
- Returning read data that exceeds the AXI data width of the remote die is treated as a violation.
- For a Read Request received from the remote die, if ARSIZE is 5 or less, it returns a Read Data message with DLEN = 2'b00; if ARSIZE is 6, it returns a Read Data message with DLEN = 2'b01; if ARSIZE is 7, it returns a Read Data message with DLEN = 2'b10; Accordingly, if the local die receives a read data message from the remote die whose size is smaller than its own AXI data width, it duplicates the data to match its own AXI data width.

## 2.3.2. Architecture

This section describes the block diagram of AOU_CORE.

### 2.3.2.1. Block diagram

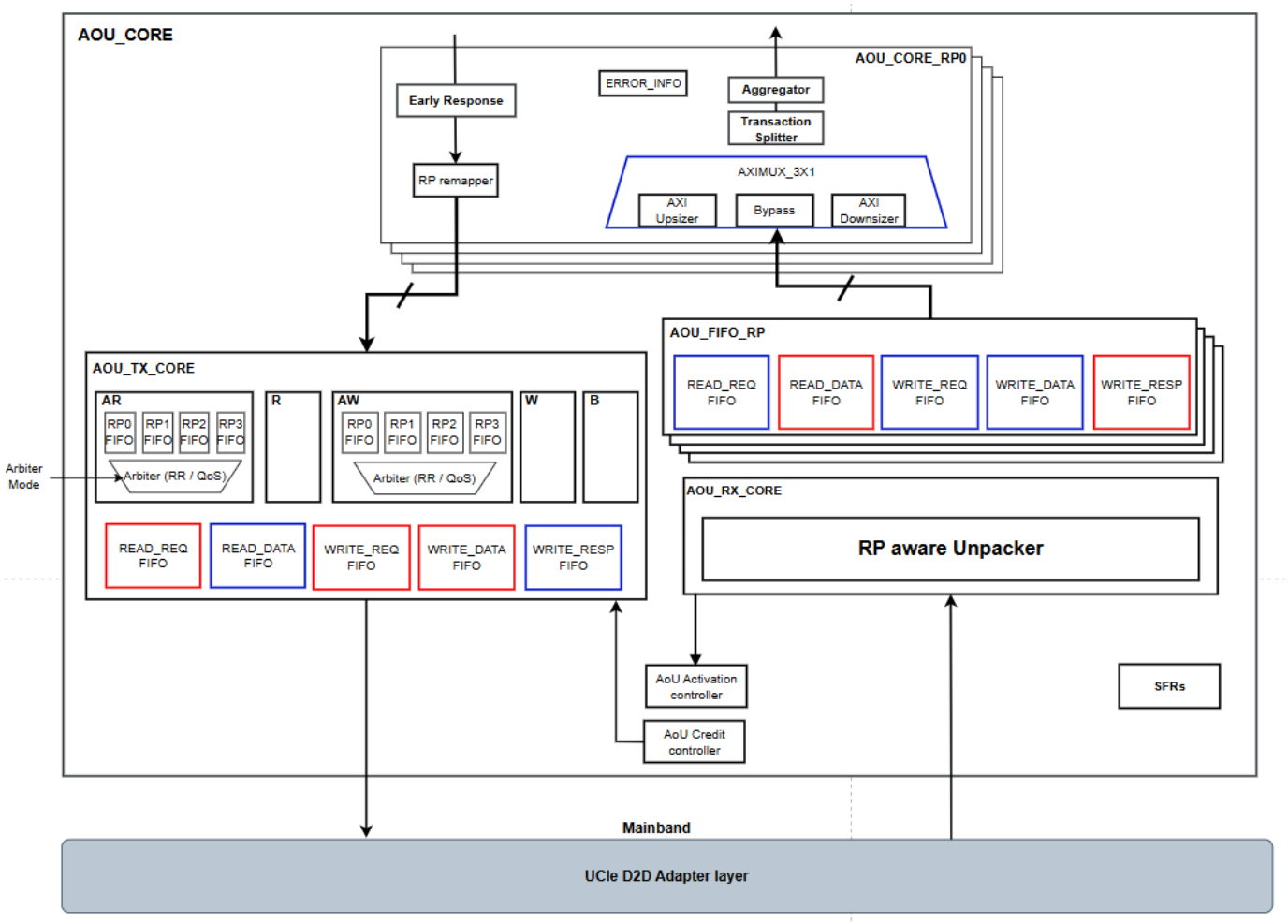


*Figure 109. Architecture of AOU_CORE*

Figure 109 shows block diagram of current AOU_CORE. The AXI interface supports both Manager and Subordinate transactions for message transmission and reception. AOU_CORE may issue AR and AW requests to the remote die using the local bus width. Consequently, R-data from the remote die and W-data from the local die are constrained to the local bus width. However, since the remote die can transmit data with variable DLEN, AOU_CORE must support all possible data lengths. This functionality is managed by AXI4MUX_3X1_TOP, which receives variable DLEN data from RX_CORE, aligns it to the local bus width, and splits the burst length into segments matching the local bus burst size.

Inside AXI4MUX_3X1_TOP, AXI aggregator is added for improving bus utilization. AXI aggregator combines narrow read & write data and send it to BUS. This functionality may modify AXI Len & Size. For example, AXI AR with 256-bit size & 16 burst length may transfer into 1024-bit size & 4 burst length. This operation can be controlled via SFR setting.

AOU_CORE provides a parameterizable FDI interface; the data-path width is selected by FDI_CONFIG (FDI_CFG_SP_32B / FDI_CFG_SP_64B / FDI_CFG_SP_128B for single-PHY, FDI_CFG_TP_32B_64B / FDI_CFG_TP_64B_128B for two-PHY). See the Integrator Guide for the full table of FDI_CONFIG values and resulting per-PHY widths.

### 2.3.2.2. TX_CORE

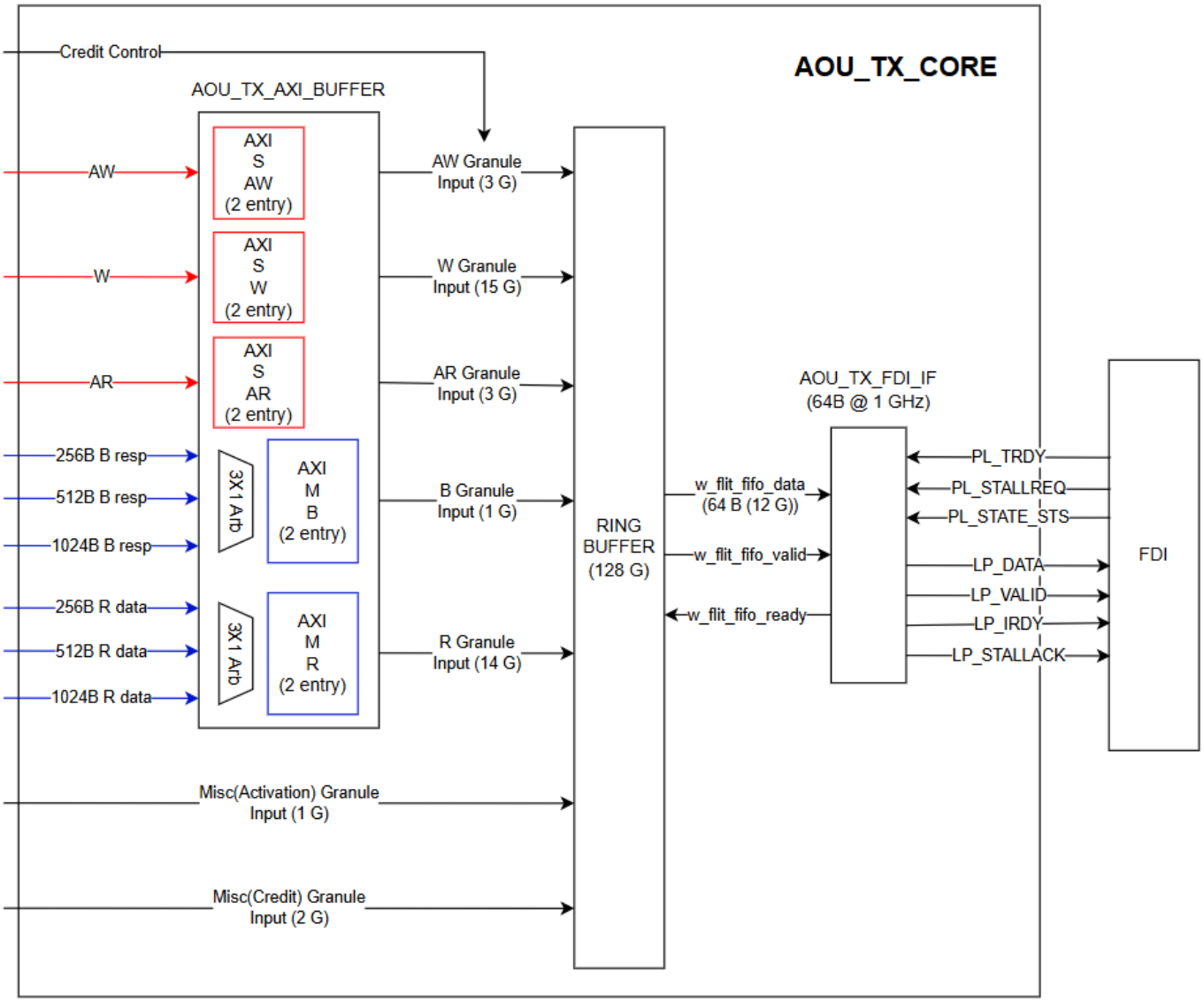


*Figure 110. Block Diagram of Single RP AOU_TX_CORE*

Figure 110 shows the simplified block diagram of single RP AOU_TX_CORE. The module receives AXI messages for Remote Die. AOU_TX_AXI_BUFFER captures AXI messages from the system bus. When AOU_TX_CORE obtains sufficient credit from AOU_CRD_CTRL, it packs the AXI message into an AOU message and stores it in the ring buffer. Once there is enough space to hold all messages, messages are stored to the ring buffer in parallel. The ring buffer subsequently performs handshake operations with the FDI interface.

**TX_CORE QoS Scheme**

Figure 111 shows the arbitration path used to select among resource-plane requests.

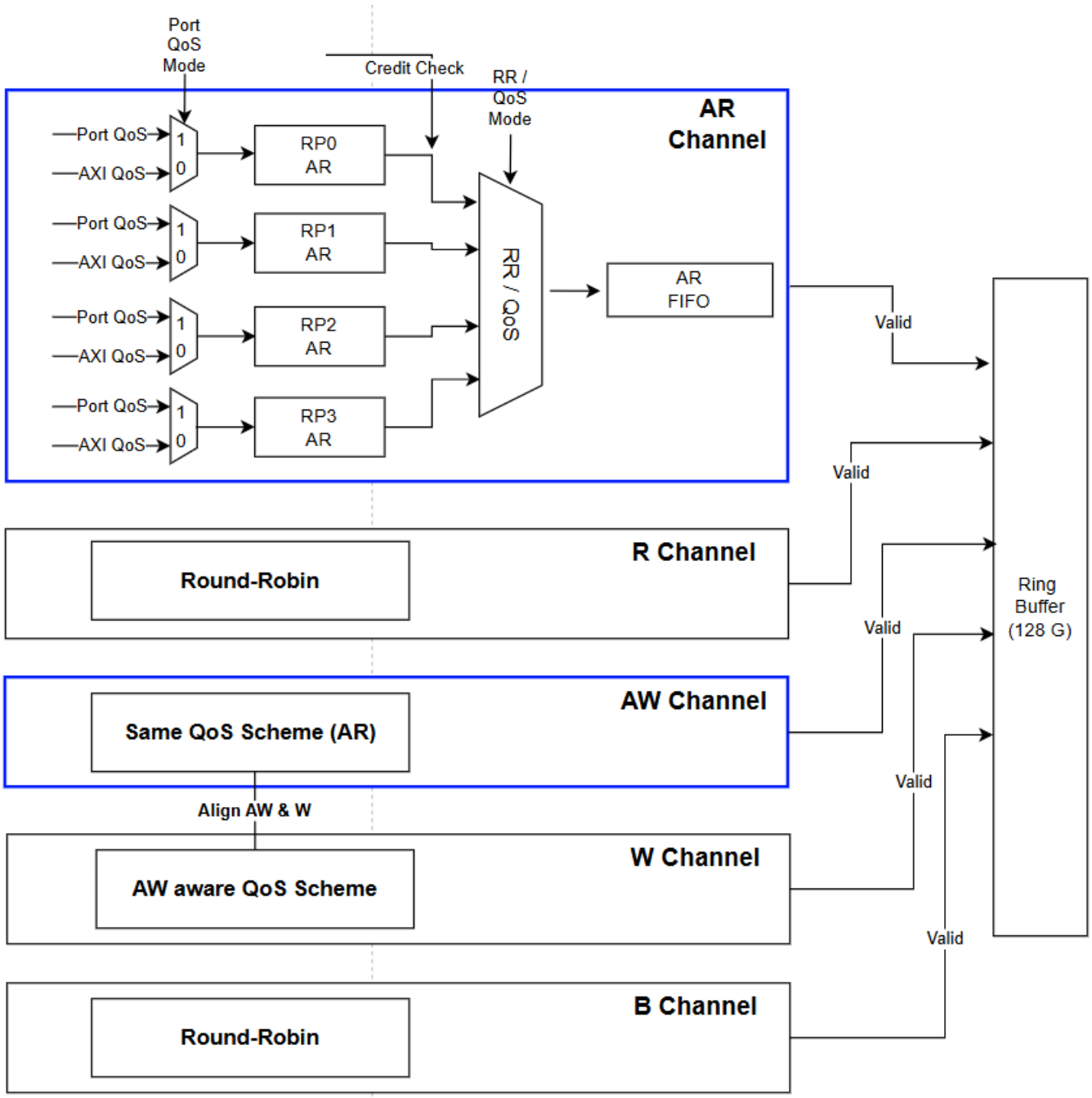


*Figure 111. TX_CORE QoS Scheme*

When multiple RP requests are issued from the TX core, a QoS scheme is applied. The TX core supports three QoS schemes: **Round-Robin**, **Port QoS**, and **AXI QoS**. Each transaction supports three QoS level: **High, Normal, Low.**

**Round-Robin (ARB_MODE = 2'b00)** : Round-robin arbitration between valid RP. When a handshake occurs, the RP that will receive the next grant is determined.

Figure 112 illustrates grant rotation among valid resource-plane requests.

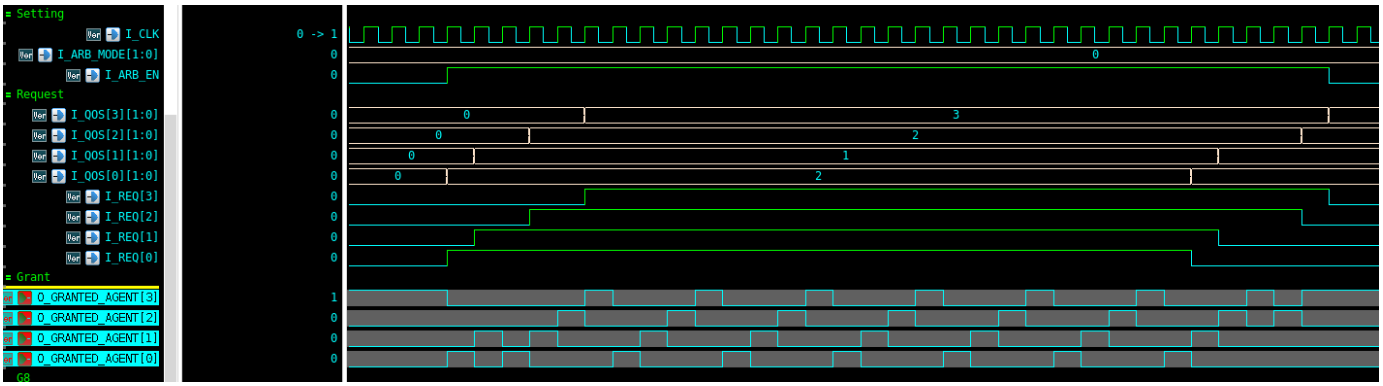


*Figure 112. Round-Robin arbitration example*

**Port QoS (ARB_MODE = 2'b01):** For each RP, the QoS level is configured by an SFR. Arbitration is performed based on the QoS value is the SFR. The AXI QoS field is not used for arbitration.

The port QoS is divided into three levels: High, Normal, and Low priority. Port QoS can be configured by setting PRIOR_RP_AXI.PRx_PRIOR. A 2-bit field is used for QoS and supports values from 1 to 3. If the field is set to 0, it is treated as 1.

By default, arbitration is performed in the order **High** → **Normal** → **Low**. If a timeout occurs while a request is at Normal or Low priority, the request is promoted and treated as High priority. Timeout value can be configured by setting PRIOR_TIMER.TIMER_THRESHOLD. The resolution used for timeout determination can be configured through PRIOR_TIMER.TIMER_RESOLUTION.

If multiple RPs have the same priority, arbitration among them is performed in a round-robin manner.

Figure 113 illustrates arbitration using the configured resource-plane priorities.

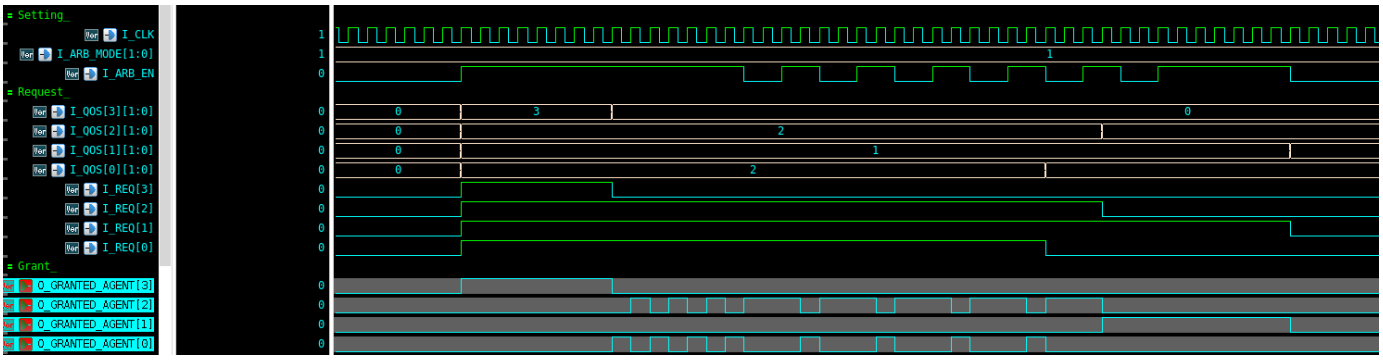


*Figure 113. Port QoS arbitration example*

**AXI QoS (ARB_MODE = 2'b10):** For each RP, the QoS level is determined by the value of the AXI QoS field. Arbitration is performed based on the AXI QoS value. For the 4-bit AXI QoS field, boundary of High priority and Normal Priority can be configured by SFR setting. The port QoS is divided into three levels: **High**, **Normal**, and **Low** priority. By default, arbitration is performed in the order **High** → **Normal** → **Low**. If a timeout occurs while a request is at Normal or Low priority, the request is promoted and treated as High priority. If multiple RPs have the same priority, arbitration among them is performed in a round-robin manner.

Figure 114 illustrates arbitration using the AXI QoS fields.

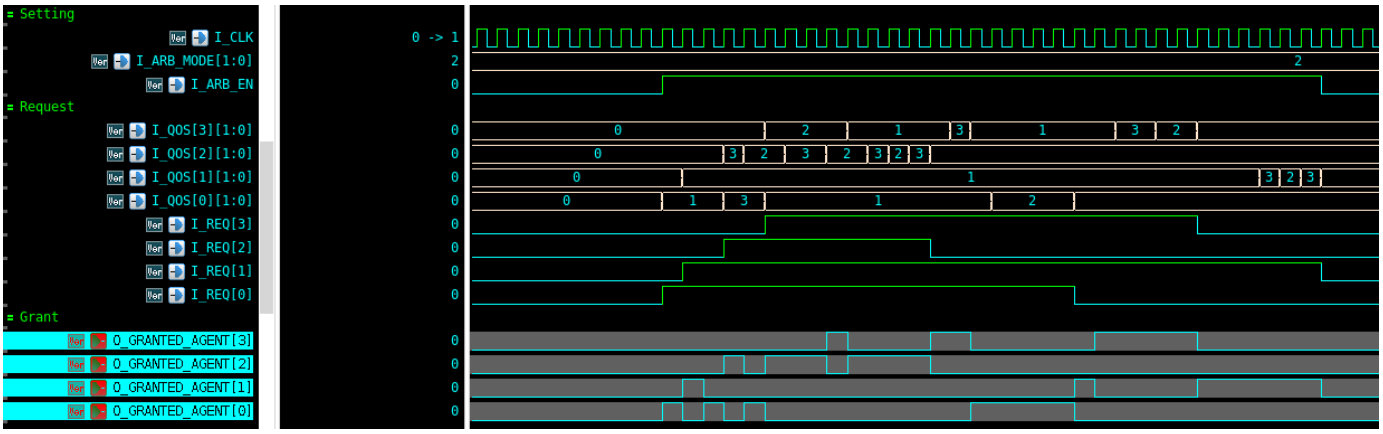


*Figure 114. AXI QoS arbitration example*

### 2.3.2.3. RX_CORE

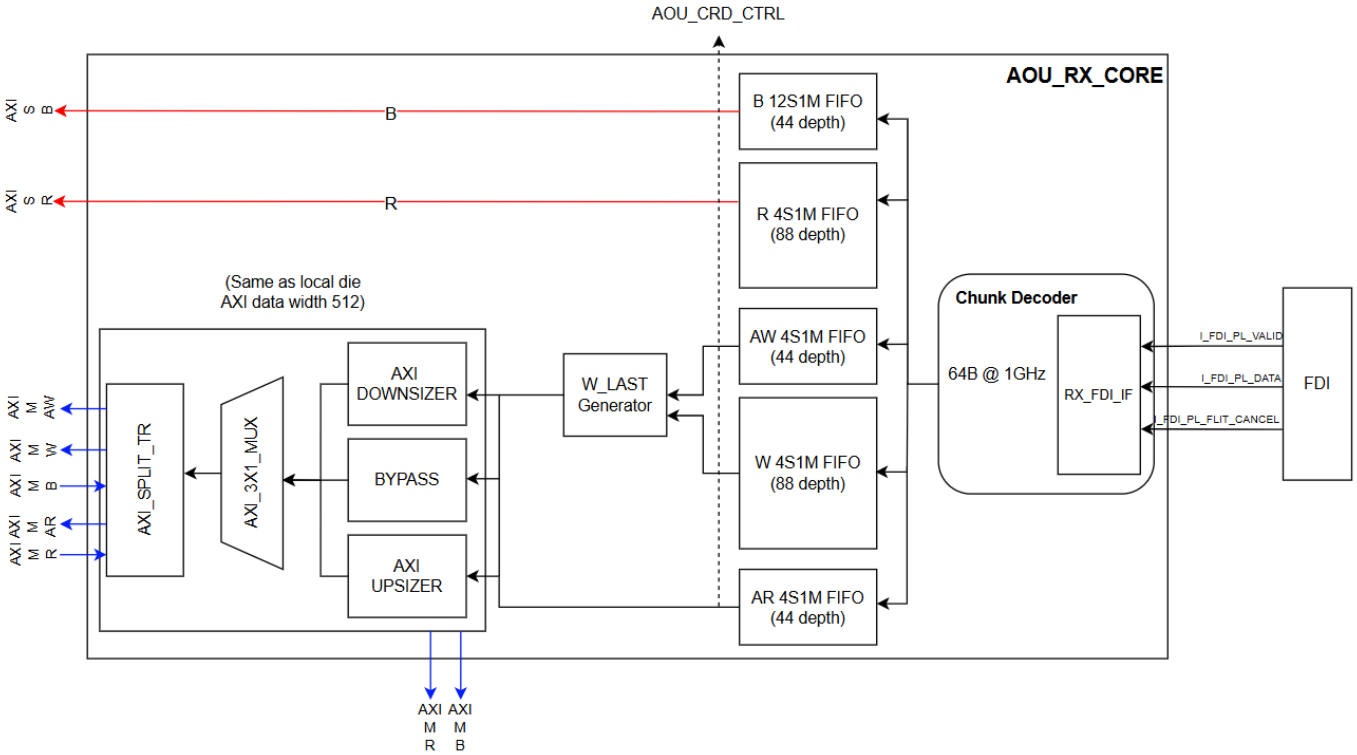


*Figure 115. Block Diagram of AOU_RX_CORE*

Figure 115 shows the simplified block diagram of AOU_RX_CORE. AOU_RX_CORE is a submodule that interfaces directly with the D2D adapter, receiving I_FDI_PL_VALID, I_FDI_PL_DATA, and I_FDI_PL_FLIT_CANCEL signals to determine the validity of incoming chunks, and unpacking messages in accordance with the AOU specification. Within RX_CORE, there is a submodule called RX_FDI_IF, which is designed to handle flit cancel as defined in the UCIe specification. This module forwards only valid chunks to the Chunk Decoder.

To ensure that the local die can support AXI transfers regardless of the AXI data width of the remote die, AOU_RX_CORE integrates both Downsizer and Upsizer modules internally.

AOU_RX_CORE includes a splitter that can divide burst lengths so that AXI messages with any burst length from the remote die can be supported on the local bus. The AW, W, and AR messages received from the remote die are sent to the AXI Manager interface, while the B and R messages received from the remote die are directed to the AXI Subordinate

interface. The R data may be smaller than the local die AXI data width depending on the ARSIZE of the corresponding AR requests, but R 4M1S FIFO duplicate values so that it is delivered to the AXI Subordinate interface with the same width as the local die AXI data width. Furthermore, the B responses corresponding to the AW and W AXI transfers issued through the Manager interface are routed into the AXI_3X1_MUX before being sent to the AOU_TX_CORE.

As illustrated in the figure above, the granularity of data transmitted from the FIFO, whether directed to the bus or to the AXI_3X1_MUX, is conveyed to the AOU_CRD_CTRL.

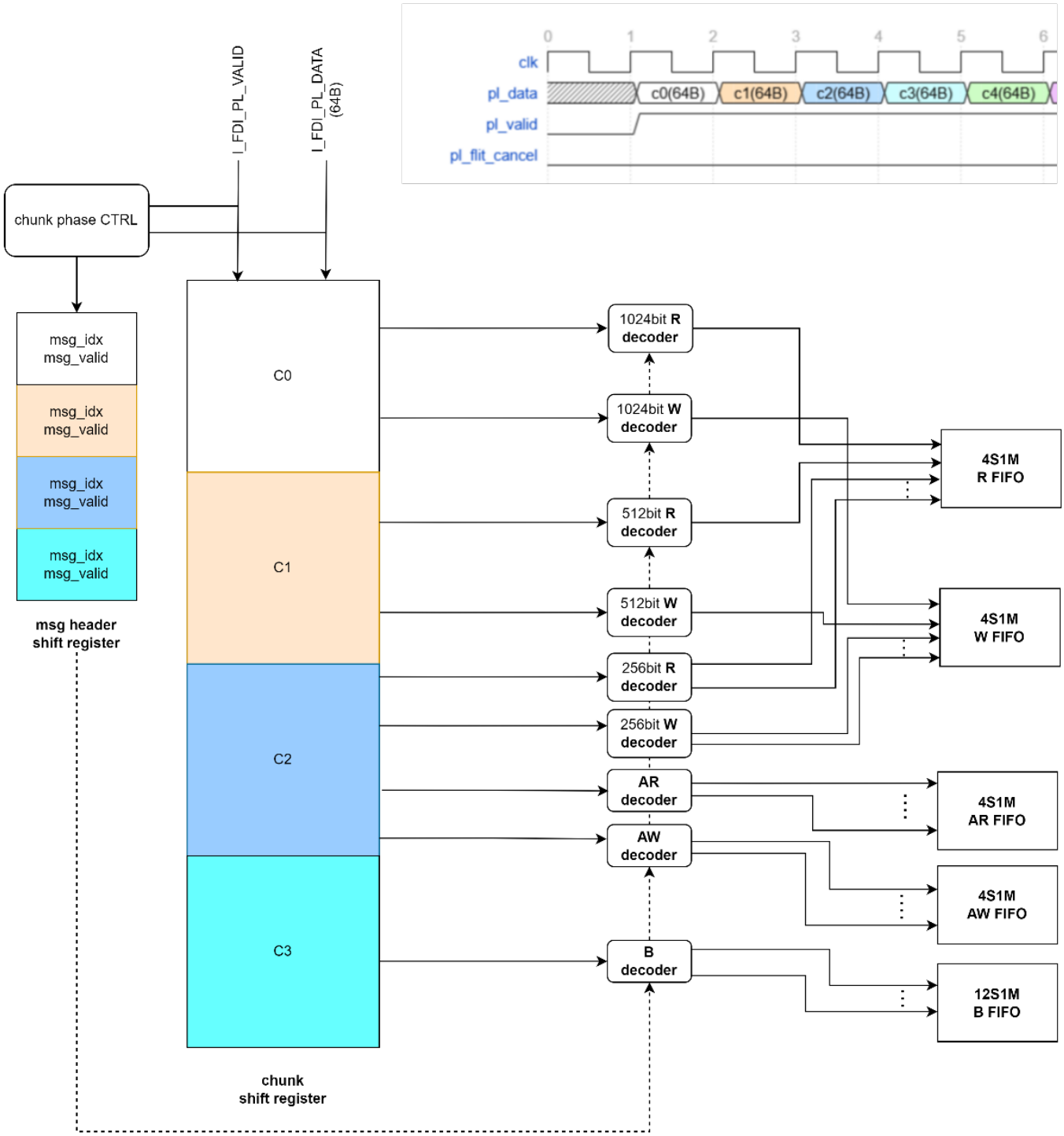


*Figure 116. Architecture of Chunk Decoder*

Figure 116 shows the architecture of the Chunk Decoder which is central submodule of RX_CORE. It consists of shift

registers and multiple decoders that decode each message, based on the received valid chunk along with its associated msg_idx and msg_valid information. Since AXI messages differ in the number of granules, the time required for decoding also varies.

For example, a B message has a size of 1 granule and can therefore be decoded immediately once a valid chunk is received from RX_FDI_IF. In contrast, Rdata or Wdata, which have a data width of 1024 bits, require 27-30 granules, and thus cannot be decoded immediately. Instead, they are decoded after approximately three cycles. Because messages of 27-30 granules may span up to four flits during transmission.

### 2.3.2.4. Message FIFO

Figure 117 shows the data FIFO organization.

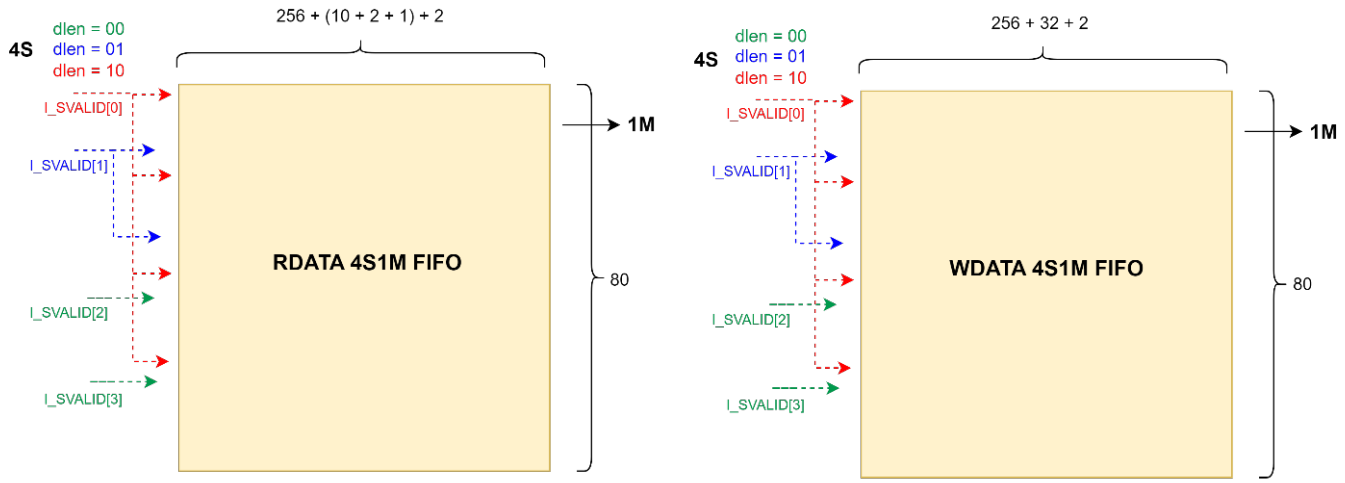


*Figure 117. Data FIFO*

Figure 118 shows the request and response FIFO organization.

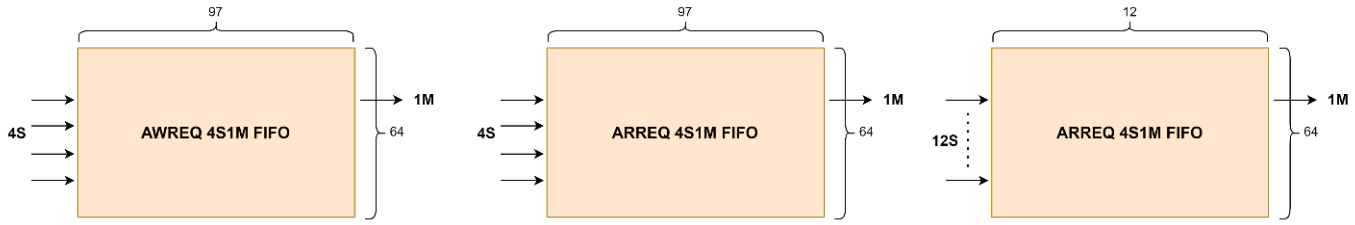


*Figure 118. Req / Resp FIFO*

The Turn-Around-Time(TAT) of AOU_CORE, including the D2D adaptor, is approximately 40 ns (equivalent to 40 cycles at 1 GHz). Therefore, the depth of the AW, AR, and B FIFOs is set to 44 regarding the margin.

The width of the Req and Resp FIFOs is equal to the length of the corresponding message. Since 12 granules are currently being decoded at once, up to 4 AW/AR requests and up to 12 B responses can be written into the FIFO in a single cycle. Therefore, the AW/AR FIFO is implemented as a 4S1M FIFO and B FIFO is implemented as a 12S1M FIFO.

In contrast, for the Wdata and Rdata FIFOs, the FIFO width and depth is defined differently.

The width of a data message varies depending on the bus data width. Therefore, the width of the Data FIFO corresponds to the message width in the case of the minimum bus data width.

For example, the width of the RDATA FIFO is 279 bits, which consists of 256 bits for the minimum bus data width, 10 bits for RID, 2 bits for RRESP, 1 bit for RLAST, and additional sdlen 2 bits to identify message width.

In aspect of WDATA FIFO, the width is 290bit. It contains 256 bits for the minimum bus data widths, 32bits for the minimum strb data widths, and additional sdlen 2bits to identify message width.

If the data received from the remote die has sdlen of 0(256bit data), the message will be stored in a single entry of the DATA FIFO. If the data received from the remote die has sdlen of 1(512bit data), the message will be stored across two entries of the Data FIFO. Similarly, if the data received from the remote die has sdlen of 2(1024bit), the message will be stored across four entries of the Data FIFO.

In addition, the depth of the Data FIFO is not 44. The depth has been determined under the assumption that the current bus data width is 512 bits. When the bus data width is 512 bits, a W message occupies 15 granules. Since the decoder in RX_CORE can decode 12 granules per cycle, the maximum W message can reach the FIFO once every 1.25 cycles (15G ÷ 12G/cycle = 1.25 cycles). Furthermore, as mentioned above, when the bus data width is 512 bits, each message is stored across two FIFO entries. Therefore, if the maximum turnaround time (TAT) to be covered is 50 ns (50 cycles), the depth can be set to 80 (50 / 1.25 x 2 = 80). Regarding the margin, Data FIFO's depth is set to 88.

When FDI_CONFIG selects a 1024b (128B) FDI interface (FDI_CFG_SP_128B or FDI_CFG_TP_64B_128B), the per-chunk granule rate roughly doubles, so the per-RP RP*_RX_W_FIFO_DEPTH / RP*_RX_R_FIFO_DEPTH parameter defaults are bumped to 140 entries to keep the same effective TAT margin. See the Integrator Guide for the parameter defaults.

FIFOs are allocated for each RP, and the depth of the FIFOs for each RP is configurable.

#### 2.3.2.5. Data width converter

**Overview**

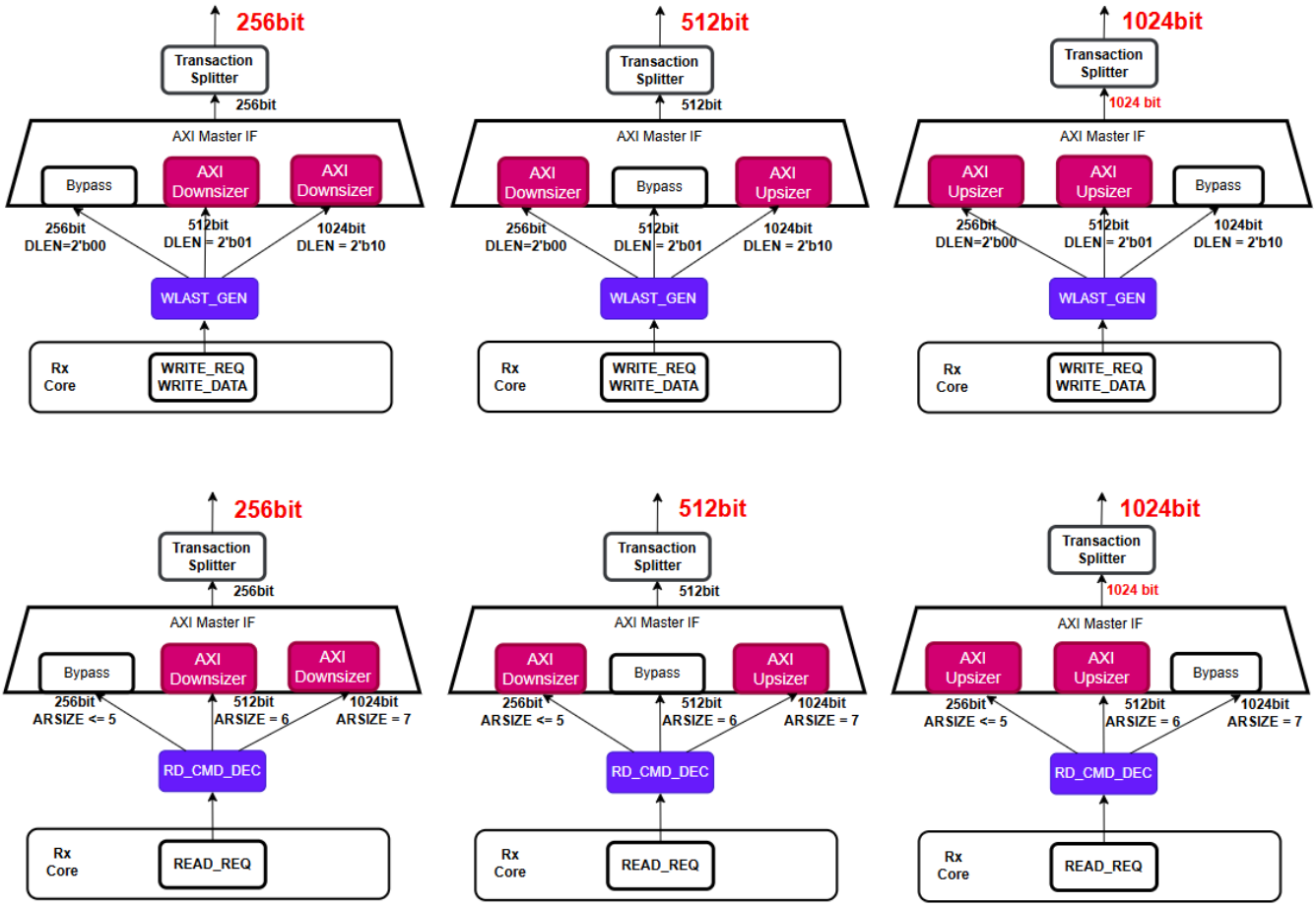


*Figure 119. AXI4MUX_3X1_TOP*

According to the AoU specification, even if the data width differs between chips, they must be able to transmit AXI transfers to each other. Because the remote die may issue read or write transactions with data widths of 256, 512, or 1024 bits, the local die is responsible for converting the data to align with its own bus data width. The response must then be reformatted to match the original request size before being returned.

Within AOU_CORE_TOP, the AOU_AXI4MUX_3X1_TOP module performs this function. Figure 119 shows how the structure of the Bypass, AXI Downsizer, and AXI Upsizer varies depending on the AXI data width of the local die.

### Width Adaptation Paths

Three slave channel slots correspond to three possible remote data widths: 256 bits, 512 bits, and 1024 bits. Which slot uses an upsizer, bypass, or downsizer depends on the local die's AXI_DATA_WD parameter. A dedicated variant of the width-adapter module is instantiated for each supported local width:

Table 2531 identifies the width-adapter variant and conversion path for each local and remote data width.

*Table 2531. AoU width-adaptation paths*

| Local AXI_DATA_WD | Variant instantiated | CH0 (256b remote) | CH1 (512b remote) | CH2 (1024b remote) |
|---|---|---|---|---|
| 256 | AOU_AXI4MUX_3X1_256 | Bypass | Downsizer | Downsizer |
| 512 | AOU_AXI4MUX_3X1_512 | Upsizer | Bypass | Downsizer |
| 1024 | AOU_AXI4MUX_3X1_1024 | Upsizer | Upsizer | Bypass |

**Write path routing:** The DLEN field embedded in the Write Data message indicates the original data width.

AOU_AXI4MUX_3X1_TOP uses this field to steer write data and strobes to the correct channel slot (and therefore through the correct width adapter).

**Read path routing:** The ARSIZE field of the Read Request message determines which channel slot handles the read response. The 3:1 mux selects the active slave R channel based on a 2-bit selector derived from ARSIZE.

The upsizer (AOU_AXI_UP) packs multiple narrow beats into a single wider beat, recalculating AWLEN/ARLEN accordingly. The downsizer (AOU_AXI_DOWN) splits a single wide beat into multiple narrower beats. Both modules regenerate WLAST internally using a beat counter that compares the current W beat count against the recalculated burst length.

#### WLAST Regeneration

The AoU Write Data message does not carry a WLAST signal. The AOU_AXI_WLAST_GEN module regenerates WLAST by aligning the phases of WriteReq and Write Data messages. It operates upstream of the width adapters in the RX-to-AXI path within AOU_CORE_RP.

The module uses I_S_WDLENGTH (derived from the DLEN field) to determine the number of W beats expected for each AW transaction:

- When AWLEN == 0 (single-beat burst), WLAST is asserted immediately on the first AW/W handshake.
- For multi-beat bursts, WLAST is asserted when the internal remaining beat counter r_m_awch_remain_beat reaches 1.

In addition to AOU_AXI_WLAST_GEN, the upsizer and downsizer modules (AOU_AXI_UP and AOU_AXI_DOWN) each contain their own WLAST generation logic to account for the changed burst length after width conversion.

### 2.3.2.6. AXI Transaction Splitter

#### Overview

In the AoU specification, the AXI LEN field is fixed at 8 bits, allowing the remote die to issue burst transactions up to the maximum supported by an 8-bit AXI LEN. However, because the local die's bus may not support the full 8-bit burst length, inbound AXI transactions must be split into bursts that the local bus can handle and then reassembled to match the original request.

Even if the burst length increases as a result of data-width conversion, the AXI transactions are always split to ensure they do not exceed the maximum burst length supported by the local bus.

The splitter (AOU_AXI_SPLIT_TR) sits between the 3:1 arbitration mux and the AXI Aggregator inside AOU_AXI4MUX_3X1_TOP. One instance exists per RP, and the maximum burst length can be configured independently for each RP through the AOU_SPLIT_TR_RPn SFRs.

> **Note:** The AOU_CON0.AXI_SPLIT_TR_EN field exists in the SFR but is currently tied to 1'b0 in AOU_CORE_RP, making the splitter always active. The enable bit is reserved for future use.

#### Burst Splitting

The splitter uses internal address-generator submodules (aou_axi_split_addrgen_ar for reads, aou_axi_split_addrgen_aw for writes) to decompose a single long burst into multiple sub-bursts, each no longer than MAX_AxBURSTLEN + 1 beats.

Addresses for each sub-burst are computed from the original base address, burst type, and size.

The MAX_AxBURSTLEN value must be of the form 2^n - 1. Legal values are:

Table 2532 lists the supported maximum burst-length settings and resulting beat counts.

*Table 2532. AoU burst-splitting limits*

| MAX_AxBURSTLEN | Max beats per sub-burst |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 3 | 4 |
| 7 | 8 |
| 15 | 16 |
| 31 | 32 |
| 63 | 64 |
| 127 | 128 |
| 255 | 256 (no split) |

**ID width:** The splitter reserves 2 bits from the AXI ID for internal sub-burst tracking. Master-side IDs are therefore ID_WD - 2 bits wide. The 2 extra bits allow the splitter to correlate responses from up to 4 outstanding sub-bursts per original transaction.

#### Response Reassembly

**Read path:** The aou_split_tr_arch reorder buffer tracks the sub-burst sequence for each outstanding read. Slave-side RLAST is asserted only when the final beat of the last sub-burst is received, reconstructing the original burst boundary. Intermediate RLAST signals from the master side are consumed internally.

**Write path:** B responses from sub-bursts are merged. The splitter tracks outstanding write sub-bursts with r_wr_pending_cnt and returns a single B response to the slave side once all sub-bursts have completed. If any sub-burst returns an error (|I_M_BRESP_RS), the merged BRESP_ERR flag is set, propagating the error to the slave-side response.

**ID mismatch detection:** The splitter exports DEST_TABLE_RID_ERR and DEST_TABLE_BID_ERR signals that flag responses with IDs not matching any outstanding transaction. These feed the INT_MI0_ID_MISMATCH interrupt source via AOU_AXI4MUX_3X1_TOP.

#### SFR Configuration

Table 2533 identifies the SFR fields associated with burst splitting.

*Table 2533. AoU burst-splitter SFR controls*

| SFR | Field | Description |
|---|---|---|
| AOU_CON0 | AXI_SPLIT_TR_EN | Global enable (reserved, tied inactive in current RTL) |
| AOU_SPLIT_TR_RP0 | MAX_AWBURSTLEN | Max write burst length for RP0 (must be 2^n - 1) |
| AOU_SPLIT_TR_RP0 | MAX_ARBURSTLEN | Max read burst length for RP0 (must be 2^n - 1) |
| AOU_SPLIT_TR_RP1 | MAX_AWBURSTLEN / MAX_ARBURSTLEN | Same for RP1 |
| AOU_SPLIT_TR_RP2 | MAX_AWBURSTLEN / MAX_ARBURSTLEN | Same for RP2 |

| SFR | Field | Description |
|---|---|---|
| AOU_SPLIT_TR_RP3 | MAX_AWBURSTLEN / MAX_ARBURSTLEN | Same for RP3 |

These SFRs must be configured before activation and must not be changed during normal operation.

### 2.3.2.7. Write Early Response Controller

#### Overview

The AOU_EARLY_BRESP_CTRL_AWCACHE module sits between the local AXI subordinate interface and the AOU TX datapath. One instance exists per RP and can be independently enabled or disabled via the per-RP WRITE_EARLY_RESPONSE_RPn.EARLY_BRESP_EN SFR.

When enabled, the module returns an early B response (BRESP = OKAY) to the local AXI manager as soon as both the write address (AW) and the last write data beat (W with WLAST) have been received at the AXI subordinate port -- before the write is packed into AoU messages, transmitted over FDI, and completed on the remote die. This reduces the write latency seen by the local requester at the cost of deferred error reporting.

Early response applies only to **bufferable** writes (AWCACHE[0] = 1). Non-bufferable writes (AWCACHE[0] = 0) pass through the module unchanged: the AW, W, and B channels are forwarded directly between the local AXI interface and the AOU TX datapath with no early response generated, and the actual BRESP from the remote die is returned to the local AXI manager as normal.

#### Transaction Flow -- Normal Case

Figure 120 shows the message flow for a bufferable write when the remote die returns a successful response.

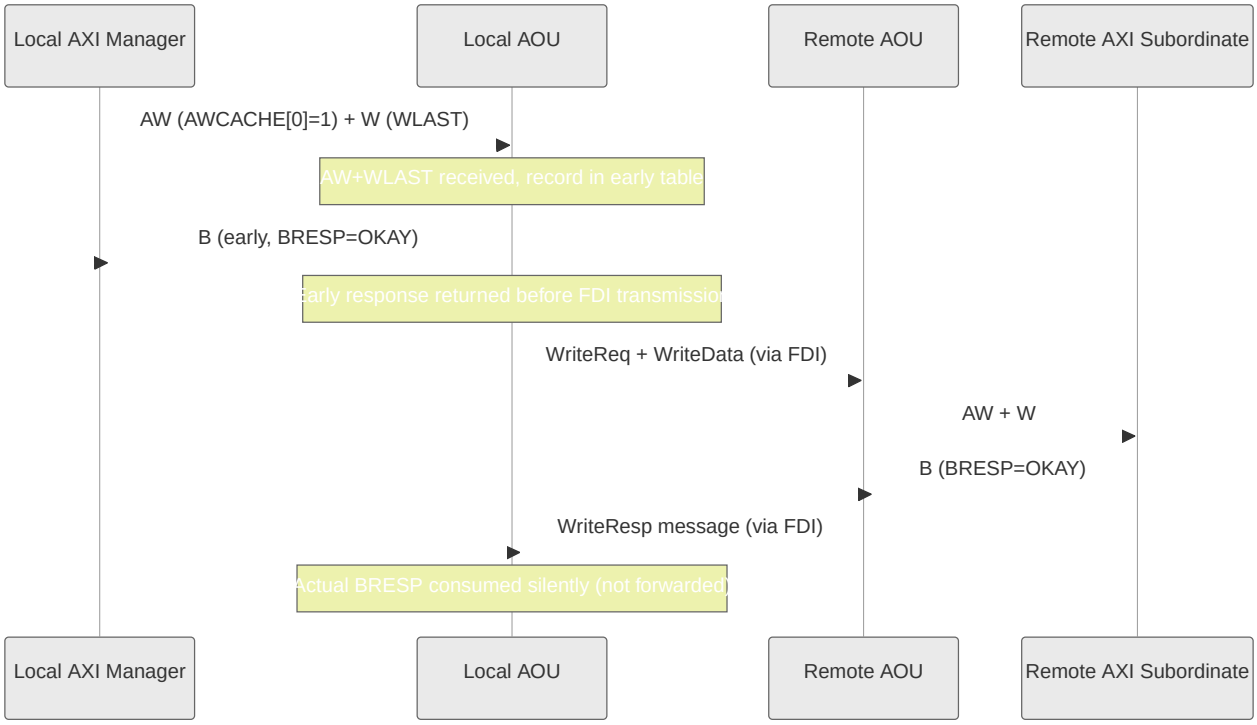


*Figure 120. AoU successful early write response*

When the actual B response arrives from the remote die for a transaction that already received an early response, the controller consumes it internally: O_AXI_S_BVALID is held low (the local AXI manager sees no second response) while

O_AXI_M_BREADY is asserted high (the response is drained from the AOU RX path).

### Transaction Flow -- Error Case

If the actual B response from the remote die carries an error (BRESP[1] = 1, indicating SLVERR or DECERR), the error cannot be reported via the AXI B channel because the early OKAY response was already returned to the local AXI manager. Instead, the error is reported via interrupt and SFR capture.

Figure 121 shows how a remote write error is consumed after the local manager has received an early response.

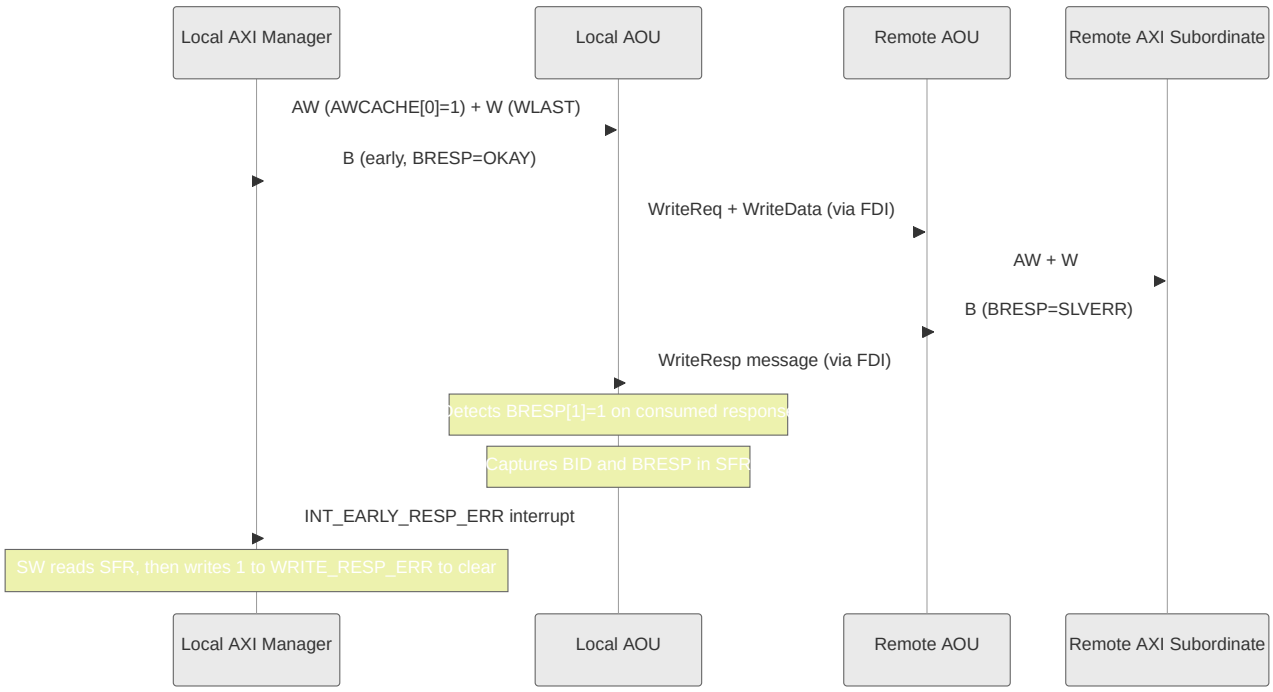


*Figure 121. AoU error after an early write response*

The interrupt fires when the consumed response has BRESP[1] = 1. SW must:

1. Read WRITE_EARLY_RESPONSE_RPn.WRITE_RESP_ERR_ID_INFO to identify the failing transaction's AXI ID.
2. Read WRITE_EARLY_RESPONSE_RPn.WRITE_RESP_ERR_TYPE_INFO to identify the error type (SLVERR or DECERR).
3. Write 1 to WRITE_EARLY_RESPONSE_RPn.WRITE_RESP_ERR (W1C) to clear the interrupt.

### ID Ordering

The AOU_EARLY_TABLE inside the controller maintains a tracking table with WR_MO_CNT entries (matching the subordinate write outstanding count). Each entry records:

Table 2534 defines the fields tracked for each outstanding write.

*Table 2534. AoU early-response tracking fields*

| Field | Purpose |
|---|---|
| id | AXI write ID (AWID) |
| bufferable | AWCACHE[0] flag |
| pending | Transaction is in flight (AW accepted, B not yet received from remote) |

| Field | Purpose |
|---|---|
| same_id_tr_cnt | Count of older pending transactions with the same AXI ID that have not yet been early-responded or actually responded |
| early_go | Set when same_id_tr_cnt reaches 0, indicating this entry is eligible for early response |
| wid_numbering | Per-ID sequence number for matching B responses to the correct table entry |

The early_go flag ensures that for transactions sharing the same AXI ID, early responses are issued strictly in order. A later bufferable transaction with the same ID waits until all prior same-ID transactions have been responded to (either via early response or actual remote BRESP). This preserves AXI ID ordering even when bufferable and non-bufferable writes with the same ID are interleaved.

#### Write-Outstanding Queue Backpressure

The backpressure mechanism described in this section is only active when the early response feature is enabled (EARLY_BRESP_EN = 1). When disabled, the module is a transparent pass-through: all AW, W, and B signals are wired directly between the subordinate and manager interfaces with no outstanding-count gating, and write depth is bounded only by whatever the downstream AOU TX datapath can accept.

When enabled, the early table has WR_MO_CNT entries (default 32, set by the S_WR_MO_CNT parameter). Every write that enters the controller -- bufferable or non-bufferable -- occupies one table entry from the time WLAST is accepted until the **actual B response** arrives from the remote die and is consumed.

Issuing an early B does **not** free the table entry. The entry remains occupied (pending = 1, out = 1) until the real remote B is received. The table depth therefore limits the total number of writes in flight across the FDI link, not just the number awaiting early response.

When r_aw_cnt == WR_MO_CNT, AWREADY and WREADY are deasserted toward the local AXI manager, stalling both the AW and W channels. The AOU TX datapath and FDI link continue operating normally during this time -- only new write acceptance is blocked. As actual B responses arrive from the remote die and table entries are freed, AWREADY and WREADY reassert and the manager may resume issuing writes.

The O_PENDING_CNT_OVER signal (exposed via the interrupt controller) fires if a write handshake is attempted while the counter is already at the limit, serving as a diagnostic for system-hang detection.

Figure 122 shows the write queue filling, stalling, and draining as tracking entries become available.

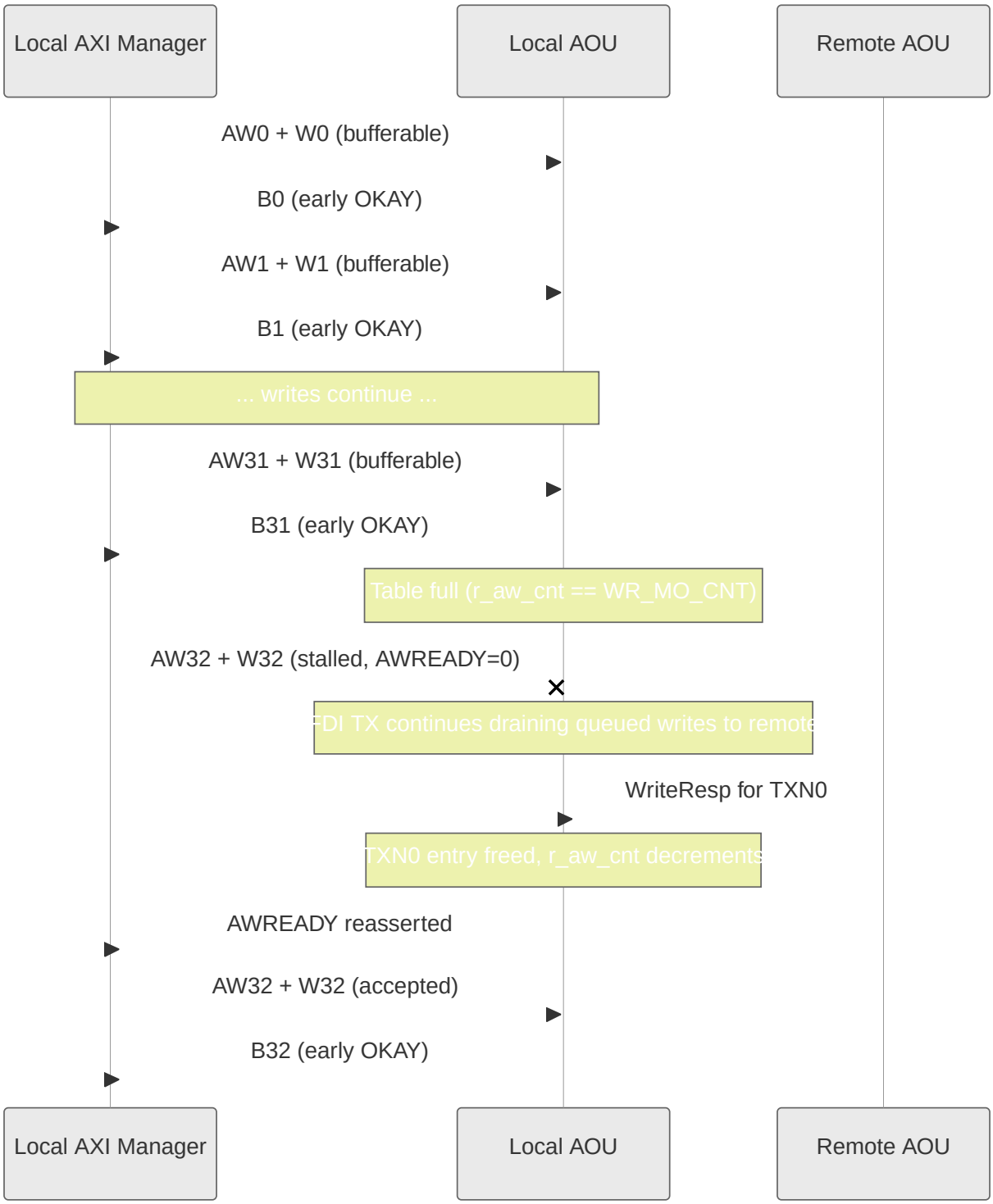


*Figure 122. AoU write-outstanding backpressure*

**Mixed Ordering -- Bufferable Before Non-Bufferable (Same ID)**

When a bufferable write (TXN0) is followed by a non-bufferable write (TXN1) on the same AWID, the controller handles them as follows:

1. TXN0 enters the table with same_id_tr_cnt = 0 and early_go = 1. Its early B is eligible immediately and is returned to the manager.
2. After TXN0's early B is issued, out is set to 1. TXN1 enters the table with bufferable = 0. The same_id_tr_cnt

computation excludes entries with out = 1, so TXN1 sees same_id_tr_cnt = 0.

3. TXN1 is non-bufferable, so no early B is generated for it regardless of early_go. It waits for the actual remote B.
4. The wid_numbering field ensures actual B responses are matched in issue order: TXN0 has wid_numbering = 0, TXN1 has wid_numbering = 1.
5. When TXN0's actual B arrives, O_EarlyResponse_Consume = 1 (bufferable entry at the read pointer), so the response is consumed silently. TXN1's wid_numbering decrements to 0.
6. When TXN1's actual B arrives, O_EarlyResponse_Consume = 0 (non-bufferable), so the response is forwarded to the subordinate as a normal BVALID.

The local AXI manager sees: early B for TXN0 first, then actual B for TXN1 -- correct AXI ID ordering is preserved.

Figure 123 shows response ordering when a bufferable write precedes a non-bufferable write on the same AXI ID.

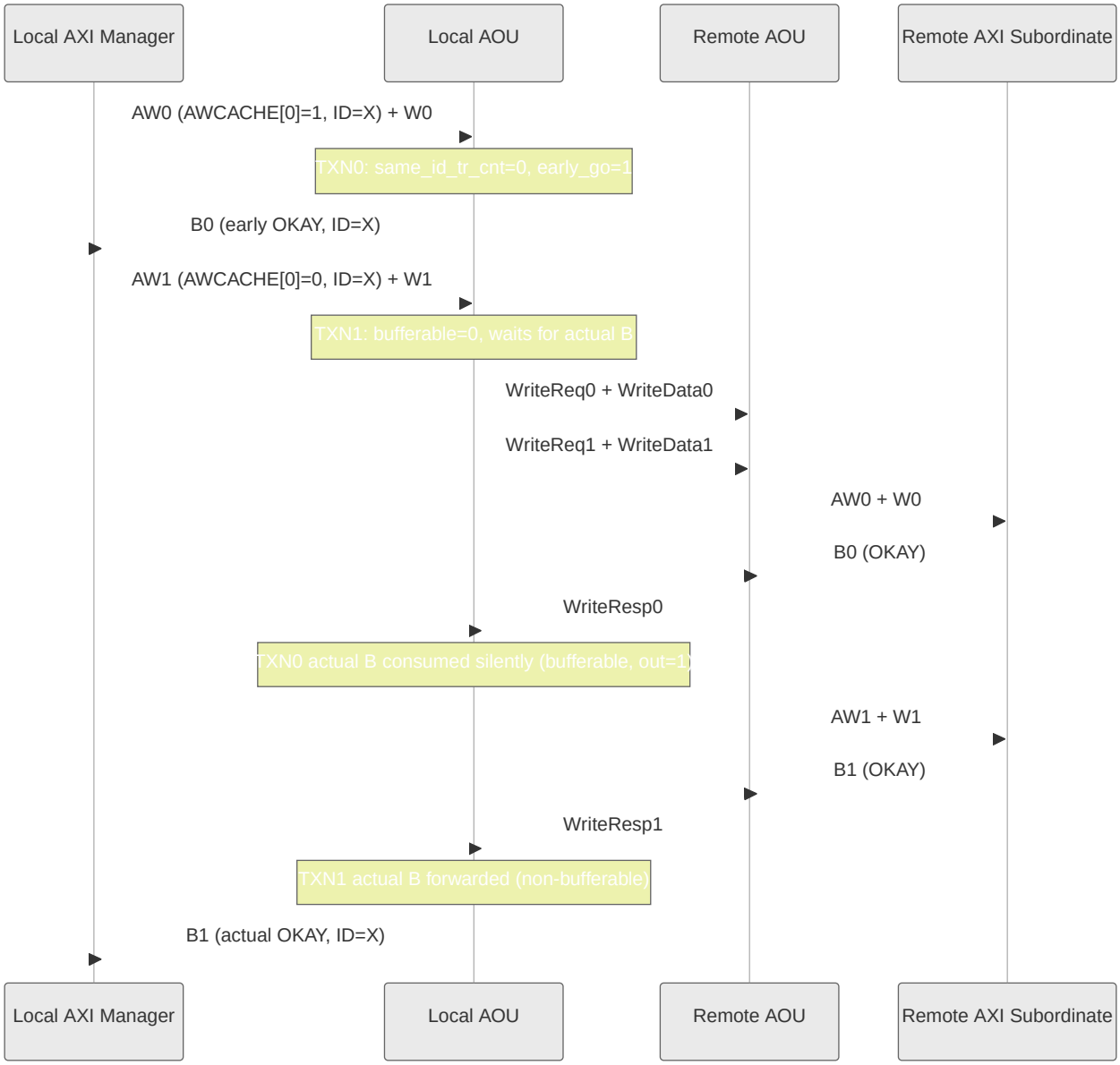


*Figure 123. AoU bufferable then non-bufferable writes*

**Mixed Ordering -- Non-Bufferable Before Bufferable (Same ID)**

When a non-bufferable write (TXN0) is followed by a bufferable write (TXN1) on the same AWID, the early response for TXN1 is **delayed** until TXN0's actual B has been received.

1. TXN0 enters the table with bufferable = 0, out = 0. No early B is generated (not bufferable).
2. TXN1 enters the table with bufferable = 1. Because TXN0 is still pending and ~out, TXN1 sees same_id_tr_cnt = 1 and early_go = 0. The early B for TXN1 is **blocked**.
3. When TXN0's actual B arrives from the remote die, it is forwarded to the subordinate (non-bufferable pass-through, O_EarlyResponse_Consume = 0). The table update loop decrements TXN1's same_id_tr_cnt from 1 to 0, setting early_go = 1.
4. On the following cycle, TXN1's early B becomes eligible and is issued to the manager.

This is the key latency impact of mixed-cacheability ordering: the bufferable write's early response is delayed until the preceding non-bufferable same-ID write fully completes its round trip through the remote die. The early-response latency benefit is lost for that transaction. Software or interconnect designers should be aware that interleaving non-bufferable writes ahead of bufferable writes on the same ID negates the early-response advantage for those bufferable transactions.

Figure 124 shows response ordering when a non-bufferable write precedes a bufferable write on the same AXI ID.

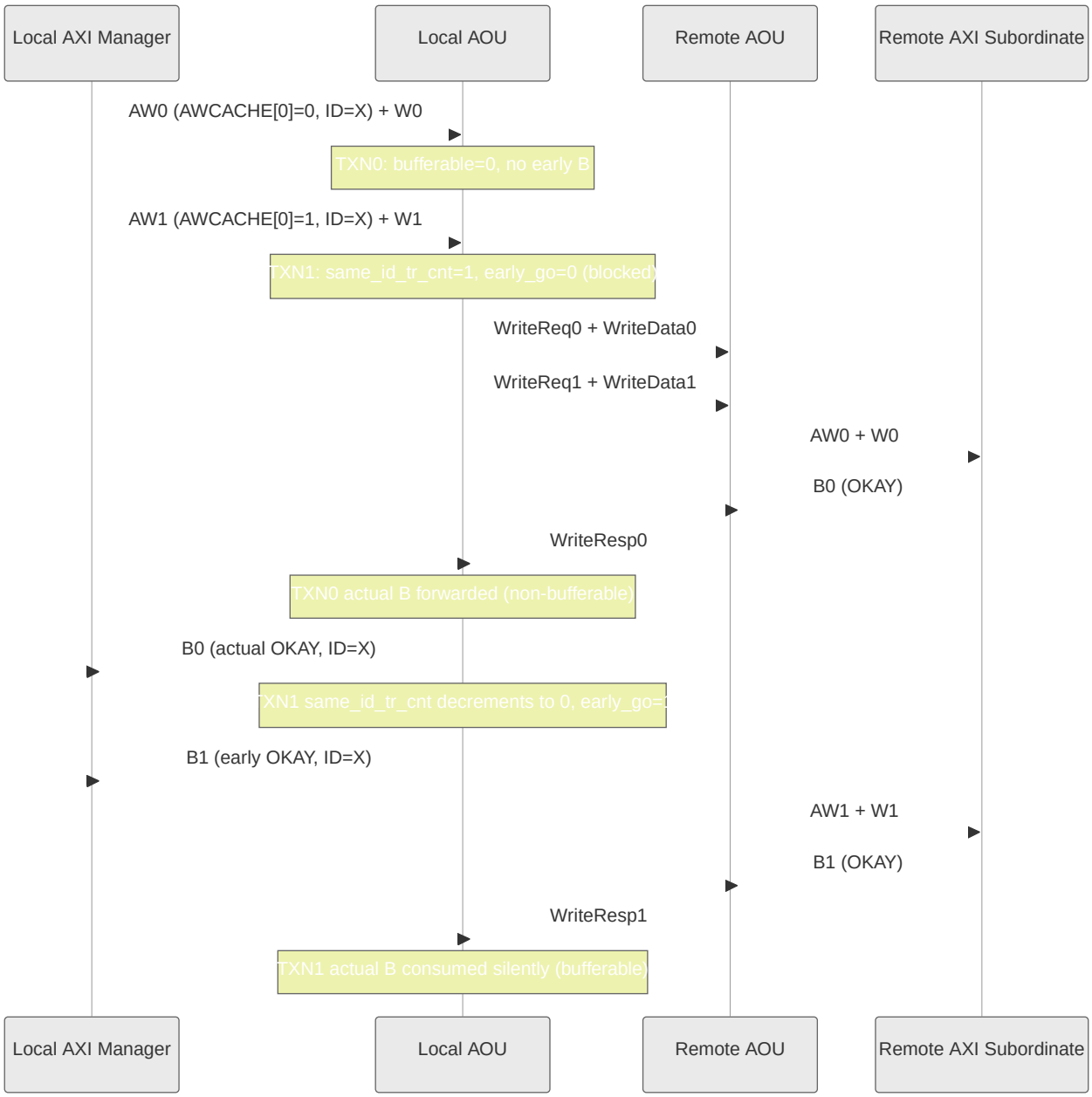


*Figure 124. AoU non-bufferable then bufferable writes*

**Enable/Disable Procedure**

The EARLY_BRESP_EN SFR may be written at any time, but the internal enable (r_early_bresp_en) only transitions when it is safe to do so:

1. Confirm WRITE_EARLY_RESPONSE_RPn.WRITE_RESP_DONE = 1 (no pending write transactions; r_aw_cnt == 0).
2. Set or clear EARLY_BRESP_EN via APB write.
3. The internal register transitions on the next clock edge where O_BRESP_DONE is asserted and no AW handshake is in progress. This guarantees that the mode change does not occur mid-transaction.

**SFR Fields**

The WRITE_EARLY_RESPONSE_RPn register (one per RP: RP0 at 0x0028, RP1 at 0x0040, RP2 at 0x0058, RP3 at 0x0070)

contains:

Table 2535 defines the per-resource-plane early-response controls.

*Table 2535. AoU early-response SFR fields*

| Field | Bits | Access | Description |
|---|---|---|---|
| EARLY_BRESP_EN | [0] | RW | Enable early B response for bufferable writes on this RP. |
| WRITE_RESP_ERR_ID_INFO | [10:1] | RO | AXI ID of the transaction whose actual BRESP had an error. |
| WRITE_RESP_ERR_TYPE_INFO | [12:11] | RO | BRESP error type (2'b10 = SLVERR, 2'b11 = DECERR). |
| WRITE_RESP_ERR | [13] | W1C | Write 1 to clear the INT_EARLY_RESP_ERR interrupt. |
| WRITE_RESP_DONE | [14] | RO | 1 when no write transactions are pending (r_aw_cnt == 0). |

See the full register map for the complete SFR listing.

### 2.3.2.8. AXI Aggregator

#### Overview

The AXI Aggregator (AOU_AGGREGATOR) combines narrow read and write data into full-width bus beats to improve bus utilization. It sits downstream of the AXI Transaction Splitter and is the last processing stage before the AXI manager port inside AOU_AXI4MUX_3X1_TOP. One instance exists per RP.

When enabled, the aggregator promotes AWSIZE/ARSIZE to the full bus width and recalculates AWLEN/ARLEN accordingly. For example, an incoming AR with size 256 bits and burst length 16 is converted to size 1024 bits and burst length 4, then issued on the AXI manager interface. When R data returns, it is split to restore the original format. The same principle applies to writes.

When disabled via the per-RP AOU_CON0.RPn_AXI_AGGREGATOR_EN SFR, all master-side signals bypass the aggregator entirely and flow directly to the AXI manager port without modification.

#### Write Aggregation

Write aggregation is activated when **both** conditions are met:

1. AWSIZE is narrower than the full bus width (AXSIZE < $clog2(DATA_WD/8)).
2. AWLEN is non-zero (multi-beat burst).

When activated, the aggregator:

- Promotes AWSIZE to the full bus width.
- Recalculates AWLEN based on the ratio between original and promoted sizes.
- OR-merges WDATA and WSTRB across consecutive narrow beats until a full-width beat is assembled, then emits the merged beat on the master W channel.
- AWBURST is passed through unchanged.

A depth-16 synchronous FIFO decouples the AW and W channels. The AW is issued on the master side only after confirming that sufficient W FIFO space is available for the recalculated AWLEN, preventing deadlocks when the downstream bus back-pressures the W channel.

Figure 125 shows narrow write beats merging into a wider downstream transfer.

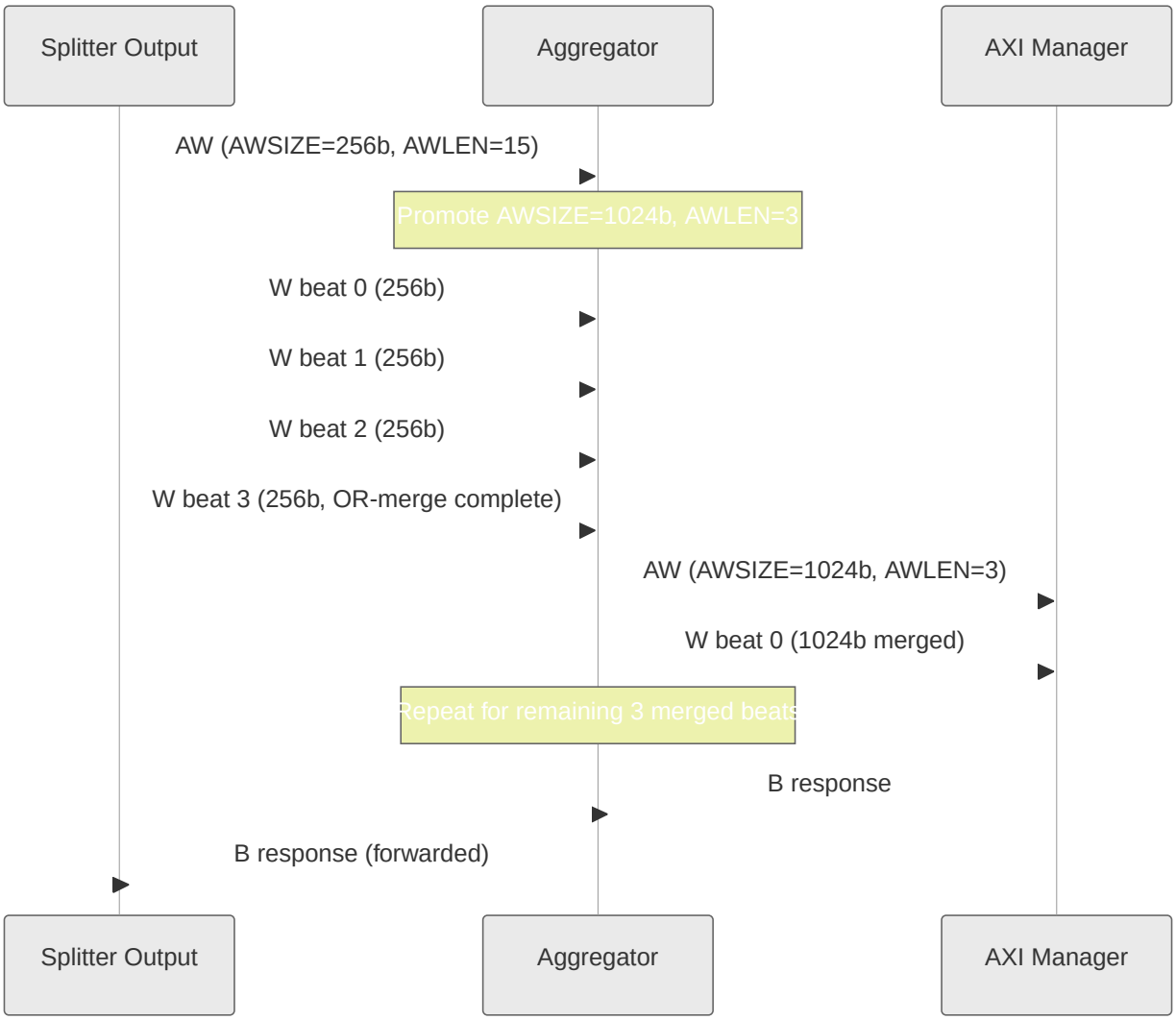


*Figure 125. Write Aggregation -- Narrow-to-Wide Beat Merging*

**Read Aggregation**

Read aggregation follows the same size-promotion logic as writes. ARSIZE and ARLEN are promoted to the full bus width, and ARBURST is passed through (the aggregator does not restrict burst type to INCR).

The AOU_AGGREGATOR_INFO submodule tracks outstanding AR transactions. For each issued AR, it records the original burst geometry (size, length) so that when wide R data returns from the manager side, the module can regenerate the correct slave-side RLAST at the original burst boundaries. Wide R beats are presented to the slave side multiple times (once per original narrow beat) with the appropriate byte-lane selection.

**SFR Fields**

Table 2536 identifies the resource-plane aggregation enables.

*Table 2536. AoU aggregation SFR fields*

| SFR | Field | Bits | Description |
|---|---|---|---|
| AOU_CON0 | RP0_AXI_AGGREGATOR_EN | [20] | Enable aggregator for RP0 |
| AOU_CON0 | RP1_AXI_AGGREGATOR_EN | [21] | Enable aggregator for RP1 |
| AOU_CON0 | RP2_AXI_AGGREGATOR_EN | [22] | Enable aggregator for RP2 |
| AOU_CON0 | RP3_AXI_AGGREGATOR_EN | [23] | Enable aggregator for RP3 |

These fields must be configured before activation and must not be changed during normal operation.

### 2.3.2.9. Credit Controller

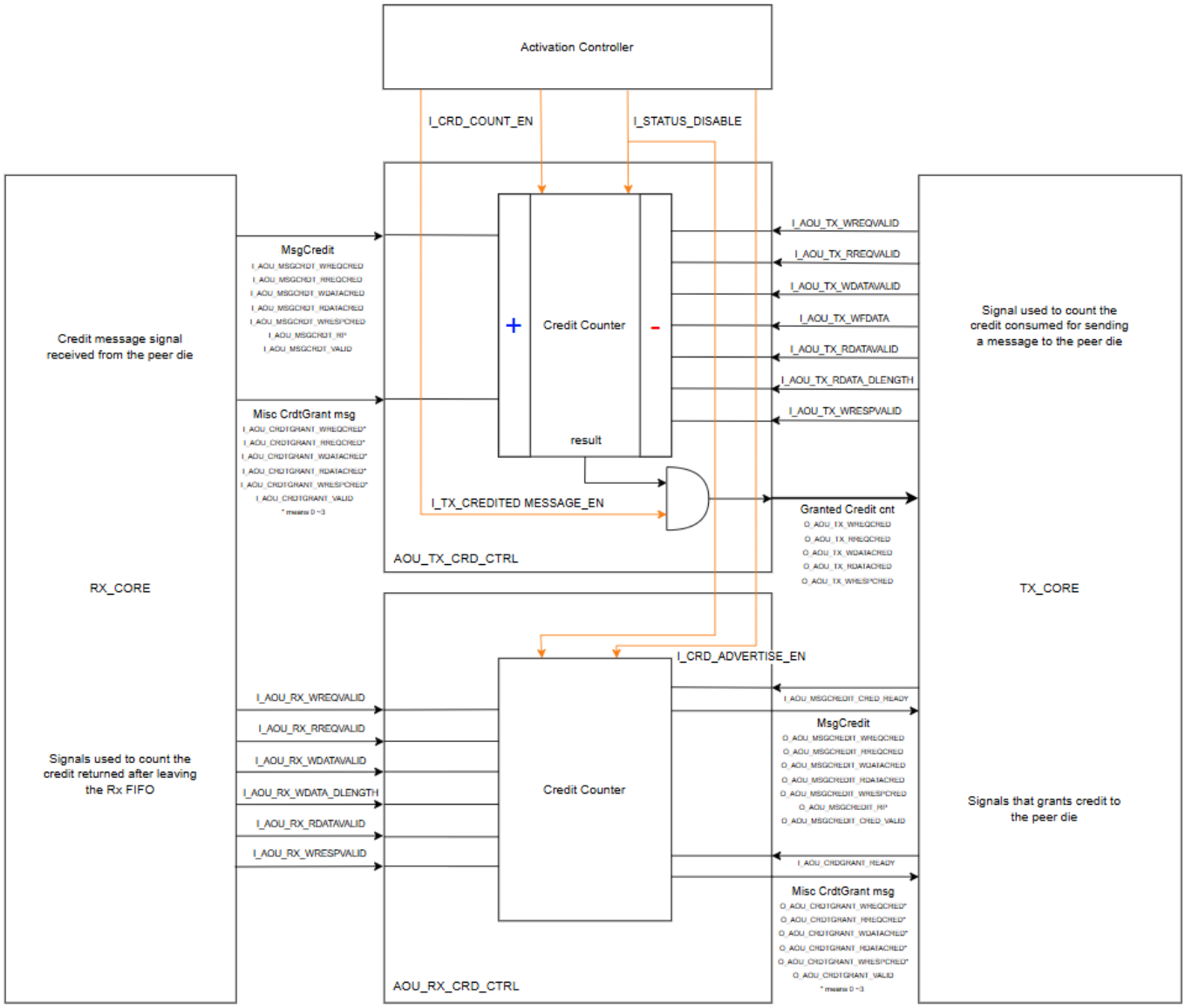


*Figure 126. Block Diagram of AOU_CRD_CTRL*

Figure 126 shows the simplified block diagram of AOU_CRD_CTRL. The Credit Controller manages both the credits granted to the remote die and the credits received from the remote die. It considers the credit consumed when the TX CORE packs an AXI message into the ring buffer, and considers the credit returned when an AXI message is popped from the RX CORE FIFO. The credit controller always grants the maximum credits available and uses only one of the two

methods - either the Misc Credit Grant messages or the Message Credit field - to grant credits.**

Rx Credit**

The local die is responsible for tracking the credits advertised to the remote die and issuing new credits once they are returned. Rx Credit accounts for both advertised and returned credits and is managed by the AOU_RX_CRD_CTRL module inside AOU_CRD_CTRL.

More specifically, Rx Credit refers to the credit provided by the local die to the remote die. The remote die uses this credit to transmit messages, which are then received and unpacked by the local Rx Core. Because the processing and unpacking occur within the Rx Core, this credit is defined as Rx Credit. The maximum available Rx Credit is determined by the depth of the Rx FIFO.

To ensure that no data is lost under flow control, credits shall not be granted when sufficient buffer space is unavailable. Therefore, the amount of credit advertised shall be determined conservatively by considering the worst-case condition. Here, the worst case is defined as the condition in which the smallest amount of credit can be advertised. Consequently, the maximum available is specified as the minimum value among all possible cases when credits are fully advertised:

```
MAX AVAILABLE CREDIT = min(max advertised credit case1, max advertised credit case2, ...,
max advertised credit caseN)
```

Assuming all PROFEXTLEN values are set to 0, the maximum advertised credit is calculated as follows: **For WREQ, RREQ, WRESPCRED**, each FIFO entry is considered to be allocated according to the granule size of each message.

```
RX AW MAX CREDIT = RX AW FIFO DEPTH x WriteReq Granules
RX AR MAX CREDIT = RX AR FIFO DEPTH x ReadReq Granules
RX B  MAX CREDIT = RX B  FIFO DEPTH x WriteResp Granules
```

**In the case of WDATA**, data widths of 256, 512 or 1024 may be received. Separate FIFOs are provided for data and for strobe, each with the same FIFO depth. Each FIFO entry is 256 bits wide for DATA and 32 bits wide for STRB, ensuring efficient utilization of the FIFO. Accordingly, 512-bit width data occupies two entries, while 1024-bit width data occupies four entries. As illustrated in Figure 126, for non-Writefull message types, WDATA is stored in the DATA FIFO and WSTRB is stored in the STRB FIFO.

Figure 127 illustrates the separate storage of write data and write strobes.

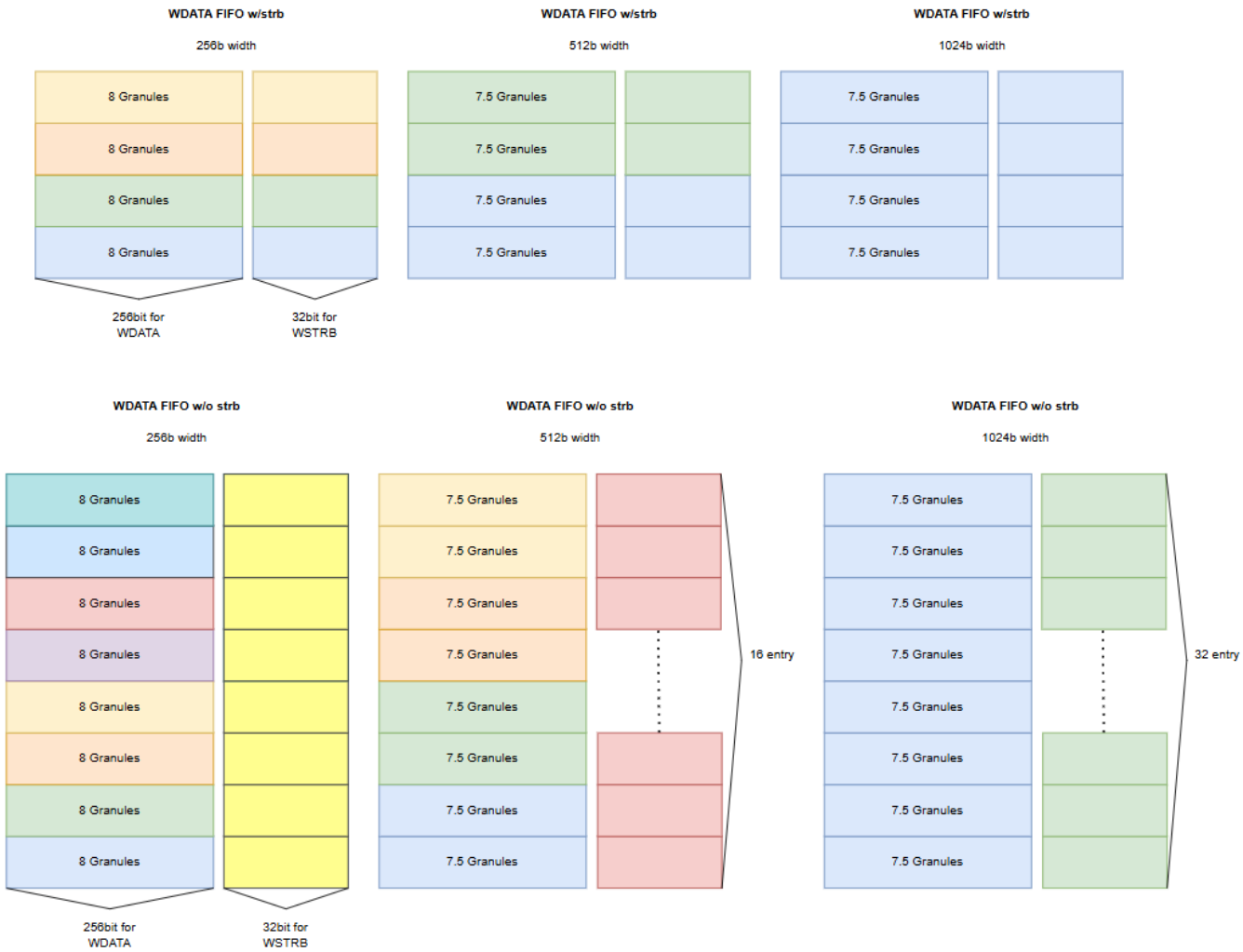


*Figure 127. Example of WDATA and WSTRB FIFO Operation*

```
Occupying Data entry per message = WDATA Width / DATA FIFO Width
```

*Table 2537. WriteData Credit Calculation*

| WDATA width | Granule size per message | Occupying Data entry per message | Granules per Data FIFO entry | Granules per Strb FIFO entry | Granules per FIFO entry |
|---|---|---|---|---|---|
| 256b | 8 | 1 | 8 | - | 8 |
| 512b | 15 | 2 | 7.5 | - | 7.5 |
| 1024b | 30 | 4 | 7.5 | - | 7.5 |

*Table 2538. WriteDataFull Credit Calculation*

| WDATA width | Granule size per message | Occupying Data entry per message | Granules per Data FIFO entry | Granules per Strb FIFO entry | Granules per FIFO entry |
|---|---|---|---|---|---|
| 256b | 7 | 1 | 7 | - | 7 |
| 512b | 14 | 2 | 7 | - | 7 |
| 1024b | 27 | 4 | 6.75 | - | 6.75 |

Table 2537 and Table 2538 illustrates all possible writedata, writedatafull message types and the credit calculation when

messages consist solely of one type. The minimum advertised credit occurs when only 1024bit writedatafull messages are received. According the maximum available credit for WDATA is defined as follows:

```
RX W MAX CREDIT = (RX W FIFO DEPTH / 4) x WriteData Full1024bit Granules
```

**In the case of RDATA,** Receiving RDATA from the remote die implies that the local die has already issued a ReadReq message. The ARSIZE used when sending is determined by the AXI_DATA_WD of the local die. For example, if AXI_DATA_WD is 512, the AR will request RDATA of 512 bits or less, and the received RDATA will therefore be 512 or 256 bits wide.

Figure 128 illustrates read-data storage at different AXI data widths.

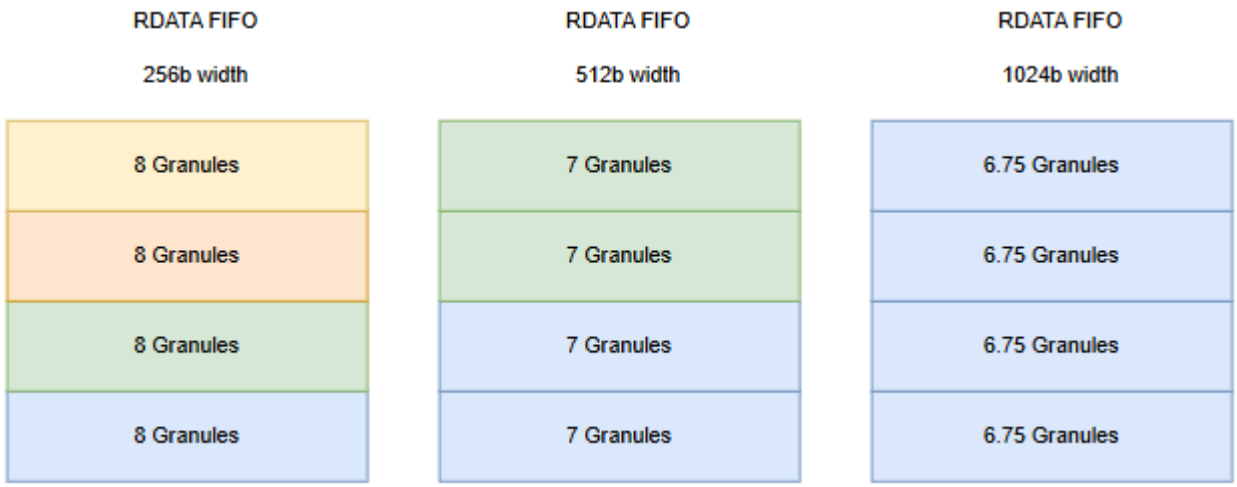


*Figure 128. Example of RDATA FIFO Operation*

Table 2539 relates read-data width to FIFO occupancy and granule credits.

*Table 2539. ReadData Credit Calculation*

| RDATA Width | Granule size | Occupying entry (RDATA width / FIFO width) | Granules per FIFO entry (Granule size / Occupying entry) |
|---|---|---|---|
| 256 | 8 | 1 | 8 |
| 512 | 14 | 2 | 7 |
| 1024 | 27 | 4 | 6.75 |

Similarly, for RDATA, credits must be advertised conservatively by considering the maximum data width associated with AXI_DATA_WD. As AXI_DATA_WD increases, the maximum credit that can be advertised decreases. Therefore, credit shall always be calculated based on AXI_DATA_WD. For example, if AXI_DATA_WD is 512, the possible RDATA widths are 256, 512, and credit must be determined under the conservative assumption that all RDATA widths are 512bits. Likewise, if AXI_DATA_WD is 1024, the possible RDATA widths are 256, 512, and 1024, and credits must be determined under the conservative assumption that all RDATA are 1024bits.

```
RX R MAX CREDIT = {RX R FIFO DEPTH / (AXI_DATA_WD / 256)} x AXI_DATA_WD RDATA Granule
Size
```

**Tx Max Credit**

- Tx Credit refers to the credit that the local die receives from the remote die. Based on this credit, the Tx Core is permitted to pack messages into flits and transmit them. For this reason, the term is defined as Tx Credit.
- Tx Credit is managed by the AOU_TX_CRD_CTRL in AOU_CRD_CTRL, which is responsible for counting the credits received from the remote die and tracking the credits consumed during message packing.
- It is recommended that Tx Max Credit parameter be set to the maximum credit value that the remote die is capable of granting.
  - If Tx Max Credit is configured to a value lower than the maximum credit issued by the remote die, credit accounting shall be limited to the Tx Max Credit value. In this case, flow control operates strictly within the range defined by Tx Max Credit, regardless of the remote die's capability.
  - If Tx Max Credit is configured to a value greater than or equal to the maximum grantable credit of the remote die, the local die can fully utilize all credits actually granted by the remote die.
  - This behavior implies that the Tx Max Credit setting is not dependent on the remote die's internal configuration. The local die shall simply make use of the credits that are actually granted to it.

**Credit Advertisement Mechanism**

Upon receipt of an ActivateReq from the remote die, AOU_CORE_TOP responds with an ActivateAck and, at the same time, advertises credit through a Misc CreditGrant Message. Only the initial credit advertisement is performed through this Misc message. Because only a single Resource Plane is implemented, after the initial advertisement, subsequent credit advertisements are carried by the Message Credit field within the protocol header. In multiple-RP configuration, credit advertisement is likewise delivered exclusively through the Message Credit field within the protocol header. Credits for multiple RPs are advertised in a round-robin manner. A grant is issued only when at least one of the credit fields - wreq, wdata, rreq, rdata, or wresp - contains a non-zero value to be advertised.

**Definition of Tx Credit Consumption and Rx Credit Advertisement/Return Timing**

- The Tx Credit is increased immediately upon receipt of a credit message from the Rx Core, within the updated value taking effect in the following cycle. Credit is decremented when the Tx Core packs a message and places it into the ring buffer. The consumption point is defined at the time of insertion into the per-RP Tx FIFO.
- The AOU_RX_CRD_CTRL exchanges credit messages (PLP protocol header MsgCredit field, dedicated Misc message) with the Tx core through valid/ready signaling. Tx core asserts CRDTGRANT_READY when the Tx core can pack a Misc Credit Message or asserts MSGCREDIT_CRED_READY when a protocol header is required to be transmitted. A credit is considered advertised at the point when the credit message handshake (valid and ready both asserted) occurs. At that instant, the advertised credits are treated as already reserved for the RX FIFO. Accordingly, a credit is not considered returned when a message leaves the RX Core, but only when data is actually dequeued from the RX FIFO, releasing the previously reserved credit.

### 2.3.2.10. RP Remapping

**Configuration Rules**

The Resource Plane (RP) operates based on the smaller RP count between the local and remote dies and follows a strict one-to-one mapping. RP assignments are configured through the **DEST_RP** SFR. The values of RP3_DEST, RP2_DEST, RP1_DEST, and RP0_DEST must be unique, and for any RP that is utilized, the configuration on the local and remote dies must be symmetrical.

This configuration must remain static and must be programmed before activation. It must not be altered during normal operation.

When the local and remote dies have different RP_COUNT values, the effective RP count is the smaller of the two.

Unused RP slots in the DEST_RP register are functionally don't-care but should be set to non-overlapping values for clean credit routing (see AOU Misconfiguration Behaviour).

Figure 129 shows how resource-plane identifiers are remapped across the link.

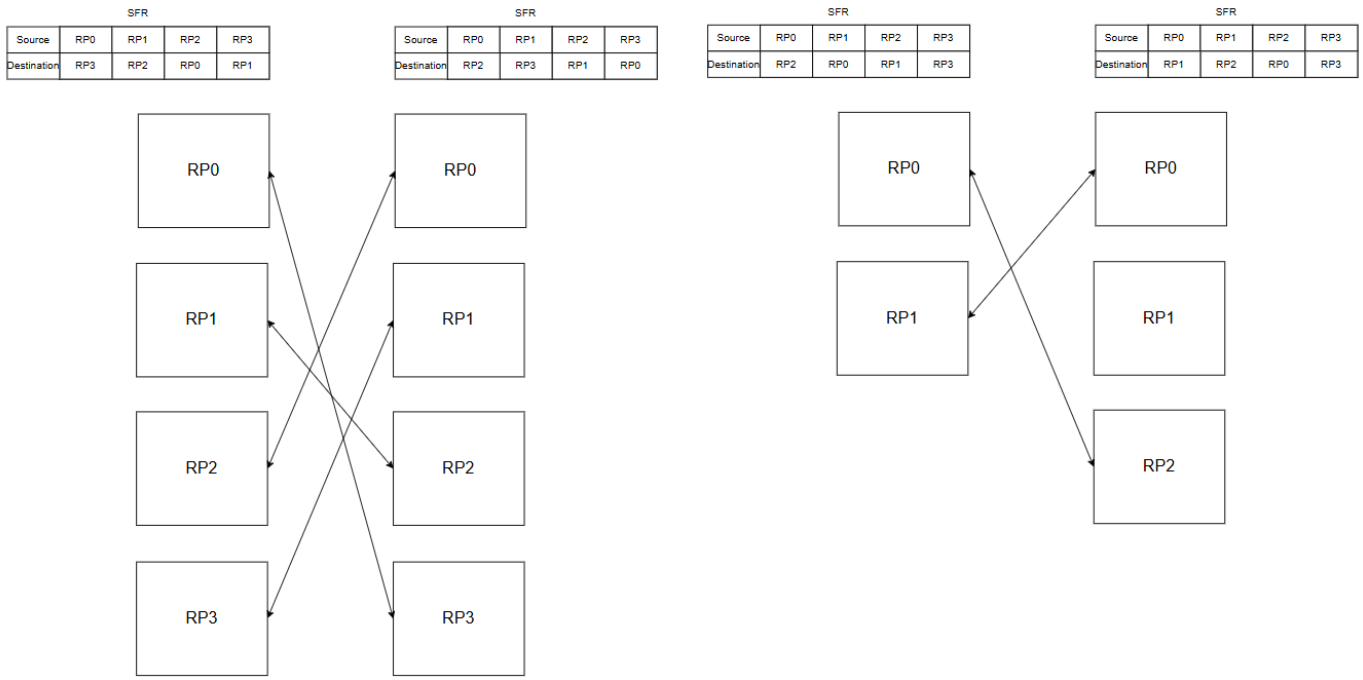


*Figure 129. Resource Plane Remapping*

**Header Encoding**

For AXI messages (WriteReq, WriteData, ReadReq), the TX path (AOU_TX_AXI_BUFFER) embeds the DEST_RP field from the SFR into the destination RP field of every outgoing message header. The remote die uses this field to route the received message to the correct RP for processing.

For credit grant messages, AOU_TX_CRD_CTRL encodes the *local* RP index (not the DEST_RP value) so that the remote die can route returned credits to the correct local RP's credit counter. This asymmetry -- destination RP in data messages, source RP in credit messages -- is fundamental to the RP remapping scheme.

**Misconfiguration Behavior**

No hardware duplicate-detection logic exists for the DEST_RP register. If DEST_RP values are configured with overlapping (duplicate) entries, the credit routing logic in AOU_TX_CRD_CTRL -- which uses a unique case statement on I_RP_DEST_RP[i] -- will fall through to the default case, mapping to the RP0 credit view. This results in silent credit accounting errors rather than a flagged interrupt or error condition.

Software is responsible for ensuring uniqueness and symmetry of the DEST_RP configuration on both link partners before initiating activation.

### 2.3.2.11. UCIe Flit cancel mechanism

AOU_CORE supports all UCIe flit cancel mechanisms. The three cases -- retransmit, flit partially valid, and full flit cancel -- are handled by the AOU_RX_FDI_IF submodule within RX_CORE.

Figure 130 illustrates the retransmit case of UCIe flit cancellation.

**Figure 10-15. Example for pl_flit_cancel for Latency-Optimized Flits and CRC Error on First Flit Half**

*Figure 130. UCIe Flit Cancel - Retransmit case*

Figure 131 illustrates the partially valid flit case.

**Figure 10-16. Example for pl_flit_cancel for Latency-Optimized Flits and CRC Error on Second Flit Half**

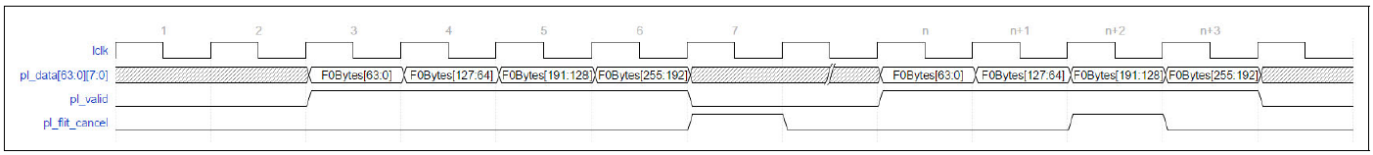


*Figure 131. UCIe Flit Cancel - Flit Partially Valid case*

Figure 132 illustrates the full flit-cancel case.

**Figure 10-17. Example for pl_flit_cancel for Latency-Optimized Flits and CRC Error on Second Flit Half, Alternate Implementation Example**

*Figure 132. UCIe Flit Cancel - Full Flit Cancel case*

### Cancel Cases

The AOU_RX_FDI_IF module uses the FDI pl_valid and pl_flit_cancel signals to determine how to handle each incoming chunk. Three cases are distinguished:

Table 2540 maps the valid and cancel signals to the receive action.

*Table 2540. AoU flit-cancel cases*

| pl_valid | pl_flit_cancel | Action |
|---|---|---|
| 1 | 0 | **Normal**: Phase counter increments. Chunk is stored in the staging buffer. |
| 1 | 1 | **Retransmit / Partial Valid**: Phase counter decrements by 1. The staging buffer is re-aligned with {zero_data, pl_data} and a valid pattern of {1'b0, 1'b1}, replacing the previously stored partial chunk with the corrected data. |
| 0 | 1 | **Full Flit Cancel**: Phase counter decrements by 2. Staging buffer entries are cleared. This is the strongest rollback, discarding the entire in-progress half-flit. |

**RX_FDI_IF Buffering**

The AOU_RX_FDI_IF module maintains a staging buffer (r_data) that is HF_PHASE_CNT entries deep, where HF_PHASE_CNT = PHASE_CNT / 2. For the default 512-bit FDI data width, PHASE_CNT = 4 and HF_PHASE_CNT = 2, so the staging buffer holds 2 chunk entries -- one half-flit of data.

A parallel valid array (r_data_valid) tracks which staging buffer entries contain valid chunk data. The output to RX_CORE is gated:

```
rx_chunk_data_valid = r_data_valid[1] && !pl_flit_cancel
```

This ensures that chunk presentation is suppressed whenever pl_flit_cancel is asserted, even if valid data exists in the staging buffer. Downstream, AOU_RX_CORE advances its phase counter (r_aou_rx_phase) only when rx_chunk_data_valid is high, so cancel-suppressed cycles stall the RX phase progression without data loss.

**Timing**

Cancel handling is purely combinational on the same clock edge as the FDI input signals. No additional pipeline stages are introduced beyond the 2-entry staging buffer and the phase FSM. The latency from FDI input to valid chunk presentation at RX_CORE is therefore determined solely by the half-flit fill time under normal operation.

> **Note:** The flit_packing_state counter (a TX-side concept in AOU_TX_CORE) is unrelated to RX cancel handling. RX cancel operates entirely within AOU_RX_FDI_IF using the phase counter r_phase.

The figures below illustrate an example of how only valid chunk data is delivered to RX_CORE. Within the design, pl_data, pl_valid, and pl_flit_cancel are defined as part of the FDI interface. In contrast, rx_chunk_data and rx_chunk_data_valid represent the signals responsible for delivering valid chunks to RX_CORE.

Figure 133 shows the selection of valid chunks for RX_CORE.

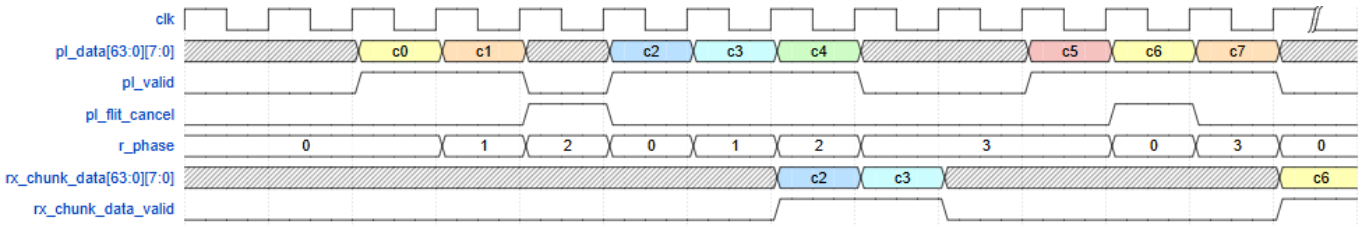


*Figure 133. Example of selecting valid chunks for RX_CORE*

## 2.3.3. Interrupts and Errors

### 2.3.3.1. Interrupt

Table 2541 lists the interrupt names, access behavior, and triggering conditions.

*Table 2541. Interrupt Sources*

| Interrupt Name | Access | Description |
|---|---|---|
| INT_ACTIVATE_START | W1C | Asserted when activation of the AoU Protocol layer is required. |
| INT_DEACTIVATE_START | W1C | Asserted when deactivation of the AOU Protocol layer is required. |
| INT_SI0_ID_MISMATCH | W1C | Asserted when AXI subordinate port received with an AXI ID response that does not match any transaction |
| INT_MI0_ID_MISMATCH | W1C | Asserted when AXI manager port received with an AXI ID response that does not match any transaction |
| INT_EARLY_RESP_ERR | W1C | Asserted when B response error occurs on early responded B response |
| INT_REQ_LINKRESET | W1C | Asserted when AOU_CORE encounters AOU SPEC protocol violation. Request link to go LinkReset. |

- **INT_ACTIVATE_START**

An interrupt that occurs when activation of the AoU Protocol layer is required. The interrupt can be cleared by writing '1' to the AOU_CORE.AOU_INIT.INT_ACTIVATE_START SFR. For details, please refer to the Interrupt description in Activation & Deactivation Flow.

- **INT_DEACTIVATE_START**

An interrupt that occurs when deactivation of the AoU Protocol layer is required. The interrupt can be cleared by writing '1' to the AOU_CORE.AOU_INIT.INT_DEACTIVATE_START SFR. For details, please refer to the Interrupt description in Activation & Deactivation Flow.

- **INT_SI0_ID_MISMATCH**

An interrupt that occurs on the AXI Subordinate Interface when a B or R channel response is received with an ID that was not previously issued as a request. For details on ID mismatch detection, see ID Ordering.

Software can check the mismatched AXI ID by reading AOU_CORE.AXI_SLV_ID_MISMATCH_ERR SFR. The interrupt can be cleared by writing '1' to AOU_CORE.AXI_SLV_ID_MISMATCH_ERR.SLV_BRESP_ERR / SLV_RRESP_ERR.

- **INT_MI0_ID_MISMATCH**

An interrupt that occurs on the AXI Manager Interface when a B or R channel response is received with an ID that was not previously issued as a request. This is detected by the Section 2.3.2.6 destination table.

Software can check the mismatched AXI ID by reading AOU_CORE.ERROR_INFO SFR. The interrupt can be cleared by writing '1' to AOU_CORE.ERROR_INFO.SPLIT_BID_MISMATCH_ERR / SPLIT_RID_MISMATCH_ERR.

- **INT_EARLY_RESP_ERR**

An interrupt that occurs when, due to the Section 2.3.2.7 feature, a B response has already been sent through the AXI Subordinate Interface, and a subsequent actual B response arrives with an error. For details, see Transaction Flow -- Error Case.

Software can check the ID and error type of the transaction in which the error occurred by reading AOU_CORE.WRITE_EARLY_RESPONSE SFR. The interrupt can be cleared by writing '1' to AOU_CORE.WRITE_EARLY_RESPONSE.WRITE_RESP_ERR.

- **INT_REQ_LINKRESET**

An interrupt that occurs when AOU_CORE receives an AoU protocol violation. Refer to LinkReset Error Reporting. When AOU_CORE receives a protocol violation, software must perform a software reset of AOU_CORE and re-enter the activation sequence.

#### 2.3.3.2. Error

**TBD**

### 2.3.4. PPA and Appendices

#### 2.3.4.1. PPA

**Performance**

Performance evaluation is conducted in both 512-bit local die and 512-bit remote die environments. The initial latency is measured from the AXI input of the local AOU_CORE to the AXI output of the remote AOU_CORE.

Table 2542 lists the measured latency of each AXI channel in a back-to-back connection.

*Table 2542. AOU_CORE Initial Latency (Cycles)*

| AXI Channel | AoU Latency on B2B connection |
|---|---|
| Read Request (AR) | 9 |
| Read Data (R) | 11 |
| Write Request (AW) | 12 |
| Write Data (W) | 12 |
| Write Response (B) | 10 |
| Misc | 9 |

The reason of two additional latency for AW and W CH is from WLAST signal generator, and data width converter.

Figure 134 shows the initial transfer latency when AOU_CORE is connected through UCIe.

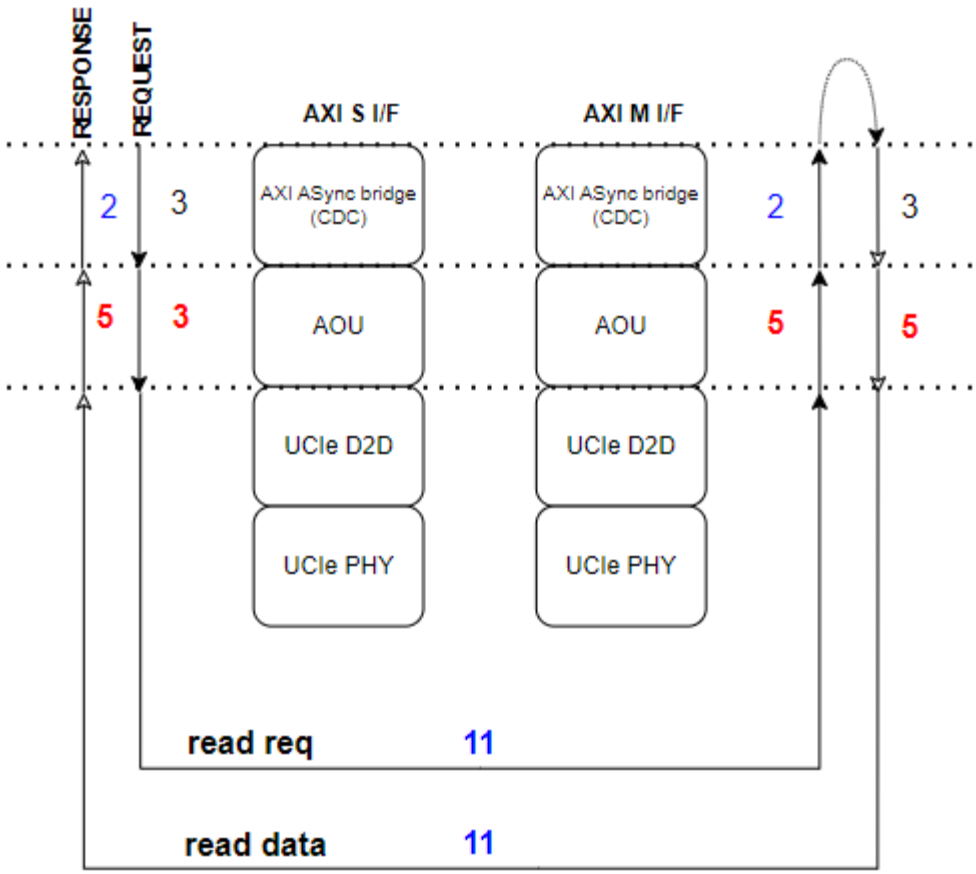


*Figure 134. Initial Latency of AOU_CORE with UCIe (Cycle)*

Figure 135 shows how bandwidth efficiency varies with UCIe FDI latency.

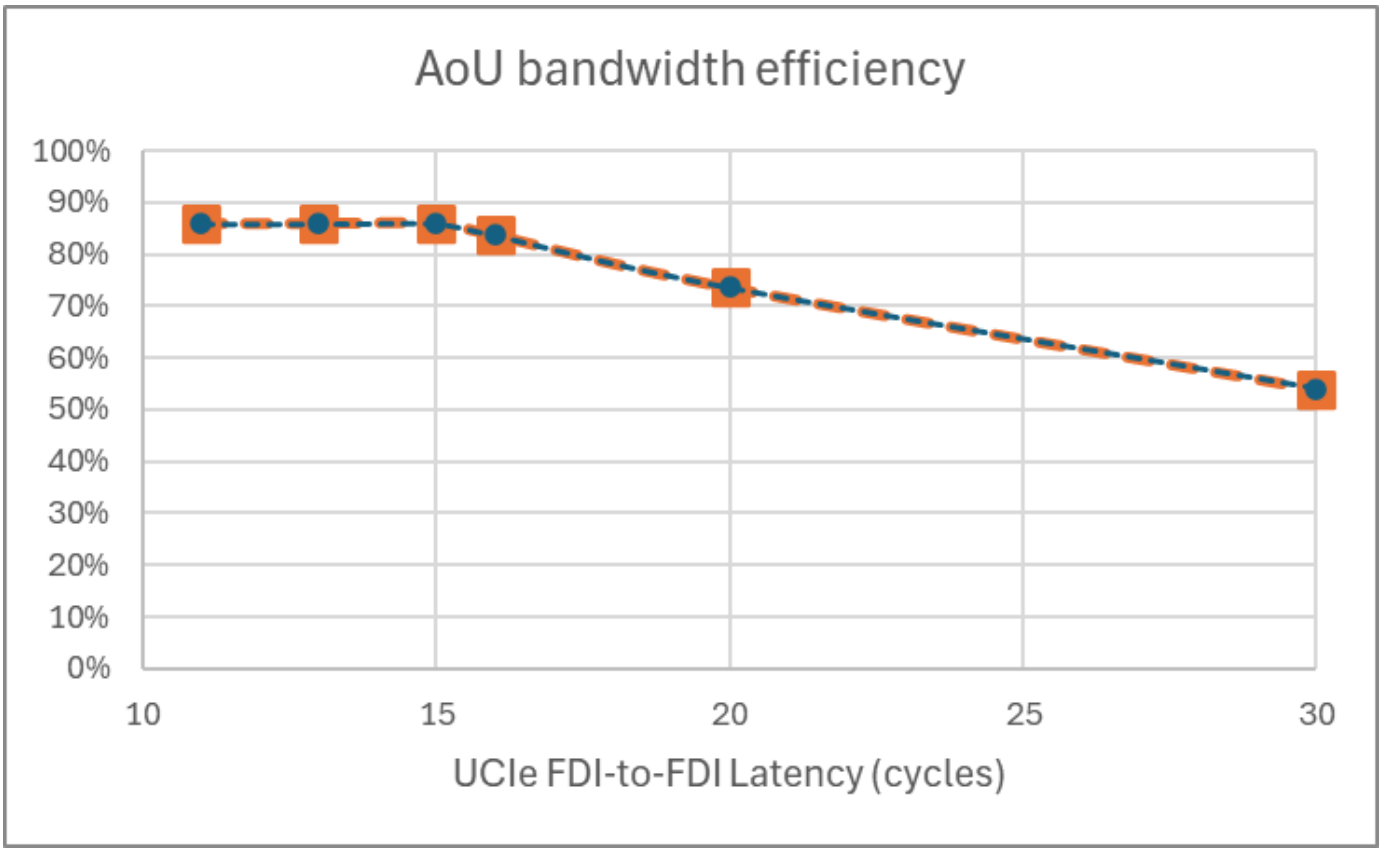


*Figure 135. AoU Bandwidth Efficiency with Respect to UCIe FDI Latency*

The results of efficiency measurements for read and write operations with varying burst lengths are presented. For write operations, two cases were evaluated. Normal Write and Write Full, where no write strobe bits are transmitted. The measured values and the corresponding expected values are as follows.

Table 2543 compares measured and expected efficiency across burst lengths and write modes.

*Table 2543. AOU_CORE Data Transfer Efficiency*

| Data transfer efficiency | | Uni-direction transfer | | Bi-direction transfer | |
|---|---|---|---|---|---|
| AXI burst length | Data type | Measured | AoU limit by spec | Measured | AoU limit by spec |
| 16 burst | Read data | 85.3%* | 85.7% | 85.3% | 84.6% |
| | Write data | 79.0% | 79.0% | 78.5% | 78.7% |
| | WriteFull data** | 84.2% | 84.6% | 84.2% | 84.2% |
| 1 Burst | Read data | 85.3% | 85.7% | 71.9% | 70.6% |
| | Write data | 66.7% | 66.7% | 63.3% | 63.1% |
| | WriteFull data | 70.3% | 70.6% | 67.0% | 66.7% |

- 0.4% performance drop is not from IP, but just from transfer data size. If the transfer data size is the multiples of 6 burst such 132 beats, it shows the same performance number with the limit in AoU spec. Everything is the same case. Note that the efficiency is measured with 128/(cycle_from_1st_valid_to_last_valid + 1), where 1 is the hidden bubble right before the first valid. It is indispensable since AoU data message size can not covered in a single 64B chunk, unlike local AXI bus.

**Power**

TBD

**Area**

Figure 136 shows the area contribution of each submodule in a single-resource-plane AoU.

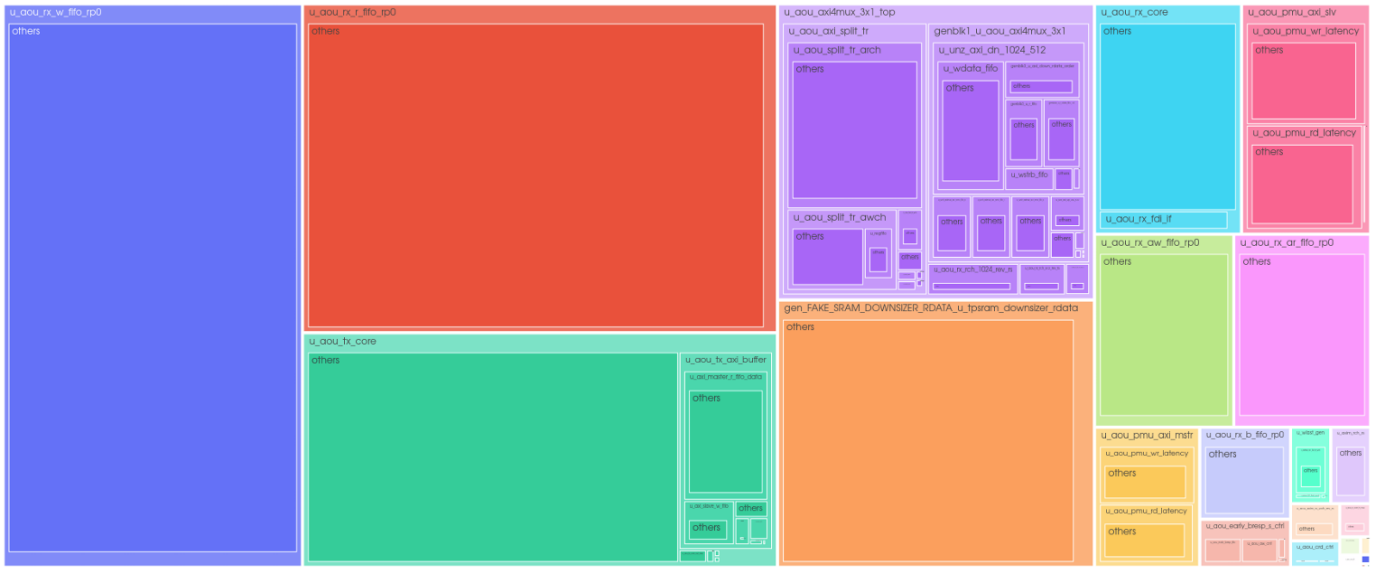


*Figure 136. Area of 1 RP AOU*

The area for each 1 RP AOU submodule is shown in the figure above.

Among them, the RX_W_FIFO and RX_R_FIFO, which store write data and read data respectively, occupy the largest portion of the AOU area. These FIFOs are currently implemented as register-based FIFOs.

FIFO Depths : TX = 2 entries / RX = 88 entries for R/W data, and 44 entries for AW/AR/B. The 88-entry default for R/W data applies when FDI_CONFIG selects a non-1024b interface; the 1024b configurations (FDI_CFG_SP_128B / FDI_CFG_TP_64B_128B) default to 140 entries, which proportionally increases the R/W FIFO area component. The Expected R/W data for 128GBs/s is 176 entries.

**v0.4 AOU_CORE**

With 2 RP, each AXI bandwidth is 512 bit and 256 bit, AOU_CORE area was 197k um^2.

The FIFO depth of the two RPs are identical.

Table 2544 compares the reported area across resource-plane configurations.

*Table 2544. AOU_CORE Area (measured at an FDI_CFG_SP_64B-equivalent configuration; numbers will scale up for FDI_CFG_*_128B configurations due to the 88 -> 140 R/W FIFO depth bump.)*

| AoU Area (um^2) | 64GB/s |
|---|---|
| 1RP | (AXI : 512bit) |
| 97K | 2RP |
| (AXI : 256bit & 512bit) | 197K |

### 2.3.4.2. Appendix B. Referenced Documents

Table 2545 lists the documents cited by the imported AoU microarchitecture specification.

*Table 2545. Referenced Documents*

| Document Name | Document Location and/or URL | Issuance Date |
|---|---|---|
| AXI-over-UCIe (AoU) Specification, Revision 0.7 | https://openchipletatlas.org | 2025 |

Register programming, activation and deactivation flows, and the debug features are in the AoU Programming Guide chapter of the Programmer's Guide.

For references from all TRM chapters, see Referenced Documents.

# Chapter 3. IP Reference

This catalogue collects the reusable IP reference pages published in the TRM. The subsystem chapters describe how these blocks are integrated into OCAH.

## 3.1. Management and I/O

- AVSBus Controller
- AXI-Lite Mailbox Unit
- GPIO
- I2C
- I3C Controller
- Log Engine
- System Timer OCTS
- Telemetry Receiver
- UART 16550

## 3.2. Security

- Deterministic Random Bit Generator (DRBG)
- eFuse
- Entropy Source
- Key Manager
- Scrambler

## 3.3. Debug and cross-trigger

- JTAG Interface Unit
- JTAG PTAP
- JTAG STAP
- Cross-Trigger Network
- Cross-Trigger Port
- Cross-Trigger Matrix

## 3.4. Interconnect infrastructure

### 3.4.1. AXI Traffic Filter

The AXI traffic filter (`axi_filter_wrap`) is the common address and attribute filter used on the inbound and outbound

AXI paths of both SEP and SMC. This section describes the decode algorithm, which is identical in every instance; the subsystem chapters describe where each instance sits in their fabric and how firmware is expected to program it.

A filter holds a configurable number of independent entries. Each entry is one `filter_ctrl` register triple — `FILTER_CONFIG`, `START_ADDR`, `END_ADDR` — and every entry is evaluated against every transaction.

#### 3.4.1.1. Instances

Table 2546 compares the filter dimensions, unmatched-traffic behavior, and bypass controls.

*Table 2546. Traffic filter instances*

| Instance | Entries | Address | Data bus | Unmatched traffic | Bypass |
|---|---|---|---|---|---|
| SEP inbound | 16 | 56-bit | 64-bit | Blocked | While SEP-scope debug is enabled |
| SEP outbound | 32 | 56-bit | 64-bit | Blocked | None |
| SMC inbound | 16 | 56-bit | 64-bit | Blocked | None |
| SMC outbound | 16 | 56-bit | 64-bit | Permitted | None |

NS and source ID matching are enabled on all four instances. Group ID matching is disabled at build time on all four, so the `group_id` field has no effect.

#### 3.4.1.2. Match, Then Permit

Decode is two distinct steps, and the difference between them governs what happens when entries overlap.

First, every entry is evaluated in parallel to produce a **match**. An entry matches only when all of the following hold:

- `entry_enabled` is 1. A disabled entry never matches.
- The transaction address falls inside the entry's address range, after the granule widening described below.
- `prot[1]` equals `allow_ns`. This is an equality test, not a permission grant: an entry with `allow_ns = 0` matches secure transactions only, and an entry with `allow_ns = 1` matches non-secure transactions only.
- The transaction's source ID equals `src_id`, or `src_id` is 0, which is a wildcard matching any source ID.
- `allow_burst` is 1, or the transaction is a single beat (`AxLEN` = 0).

Second, the **lowest-numbered** matching entry wins, and only that entry's permission bit is consulted: `read_allowed` for AR, `write_allowed` for AW. If the bit is clear, the transaction is blocked. If no entry matches at all, the outcome is the instance's unmatched-traffic policy from the Table 2546.

The two steps therefore fail differently:

- A transaction that **fails a match condition** on an entry falls through to the remaining entries, and a higher-numbered entry can still admit it. A non-secure transaction, a burst, or a foreign source ID is not denied by an entry that rejects it; it simply does not select that entry.
- A transaction that **matches** the lowest-numbered entry but finds its permission bit clear is blocked outright. Higher-numbered entries are never examined, even if one of them would have permitted the access.

Entry index is thus a strict priority and the first match ends the search, so a specific policy takes effect only at a lower index than any broader policy that overlaps it.

Writes and reads are decoded independently: the AW channel and the AR channel each run the full decode against their own address, burst length, `prot[1]` and source ID, so a write and a read to the same address may select different entries.

#### 3.4.1.3. Address Range Granule

Address comparison ignores the low address bits, so a filter range always covers a whole number of granules. `allow_burst` selects the granule:

- `allow_burst = 1`: the granule is 4 KB and address bits [11:0] are ignored.
- `allow_burst = 0`: the granule is the data bus width — 8 bytes on every instance in the Table 2546 — and address bits [2:0] are ignored.

`START_ADDR` consequently widens **down** to the base of its granule and `END_ADDR` widens **up** to the top of its granule, so a range is never narrower than one granule. Programming `START_ADDR` equal to `END_ADDR`, or any range shorter than the granule, still admits the whole granule, and a range that straddles a granule boundary admits both granules.

Table 2547 illustrates the address ranges admitted when burst filtering widens a programmed interval.

*Table 2547. Granule widening examples*

| allow_burst | START_ADDR | END_ADDR | Addresses admitted |
|---|---|---|---|
| 1 | 0x1001 | 0x1002 | 0x1000-0x1FFF, one 4 KB granule |
| 1 | 0x0FFF | 0x1000 | 0x0000-0x1FFF, two 4 KB granules |
| 1 | 0x2000 | 0x2FFF | 0x2000-0x2FFF, already granule-aligned |
| 0 | 0x1001 | 0x1002 | 0x1000-0x1007, one 8-byte granule |

Hardware writes the widened values back into `START_ADDR` and `END_ADDR` when both land in the same granule, so reading back the first example above returns `0x1000` and `0x1FFF`. When the programmed range spans more than one granule the write-back does not fire and read-back returns the programmed values unchanged, even though address matching still widens them, so read-back is not the effective range.

Widening only ever enlarges a range, so a range programmed tighter than the granule opens more address space than the programmed numbers suggest. The Programmer's Guide gives the filter programming rules.

#### 3.4.1.4. Blocked Transactions

A blocked transaction is not dropped. It is steered to an AXI error subordinate that terminates it with a `DECERR` response on `BRESP` or `RRESP`, so the initiator always receives a response.

#### 3.4.1.5. Locking a Filter Entry

Setting `FILTER_CONFIG.locked` freezes a filter entry. The bit is write-once (`woset`): once set it cannot be cleared. While it is set, every write to that entry's `FILTER_CONFIG`, `START_ADDR` and `END_ADDR` registers is steered to the same AXI error subordinate as a blocked transaction and terminates with a `DECERR` response; reads still return the locked configuration. This holds for the SMC and SEP filter instances alike.

# Appendix A: Register Field Access Notation

The register tables in this manual are generated from each block's SystemRDL description by the PeakRDL toolchain. Their **Access** column reports a field's software access together with any read or write side effect as one compact token, so a status bit that a write of 1 clears reads as `RW1C` rather than a bare `RW`. This appendix decodes that notation.

An Access cell is built in three parts:

1. a **base access** token, giving whether software can read the field, write it, or neither;
2. an optional **write side-effect suffix** joined directly to the base for an effect that an individual written bit triggers (`RW1C`, `W1S`, `RW0C`);
3. optional **space-separated flags** for whole-field write effects, read effects, and single-cycle pulses (`R RC`, `RW WS`, `W 1P`).

A field with no side effect keeps the bare base token. The same base tokens appear in the per-register summary tables, where they aggregate the access of the register's fields.

*Table 2548. Register field access tokens*

| Token | Meaning | SystemRDL property |
|---|---|---|
| **Base access** | | |
| `R` | Software can read the field. | `sw = r` |
| `W` | Software can write the field. | `sw = w` |
| `RW` | Software can read and write the field. | `sw = rw` |
| `-` | Reserved, or no software access. | `sw = na`, or a reserved gap |
| **Write side-effect suffix** — acts on the individual written bit | | |
| `1C` | Writing 1 to a bit clears it; writing 0 leaves it. | `onwrite = woclr` |
| `1S` | Writing 1 to a bit sets it. | `onwrite = woset` |
| `1T` | Writing 1 to a bit toggles it. | `onwrite = wot` |
| `0C` | Writing 0 to a bit clears it. | `onwrite = wzc` |
| `0S` | Writing 0 to a bit sets it. | `onwrite = wzs` |
| `0T` | Writing 0 to a bit toggles it. | `onwrite = wzt` |
| **Write flags** — act on any write to the field | | |
| `WC` | Any write clears the whole field. | `onwrite = wclr` |
| `WS` | Any write sets the whole field. | `onwrite = wset` |
| `WMOD` | A write invokes a block-specific hardware function. | `onwrite = wuser` |
| **Read flags** | | |
| `RC` | Reading the field clears it. | `onread = rclr` |
| `RS` | Reading the field sets it. | `onread = rset` |
| `RMOD` | Reading the field invokes a block-specific hardware function. | `onread = ruser` |
| **Pulse** | | |
| `1P` | The field self-clears the cycle after it is written, producing a one-cycle pulse. | `singlepulse` |

## A.1. References

Table 2548 follows the field-property semantics of the SystemRDL 2.0 register description language (Accellera), as

rendered by the PeakRDL toolchain.

- PeakRDL — the control and status register generator toolchain that produces these register tables.
- systemrdl-compiler field properties — the `onwrite`, `onread` and `singlepulse` properties and their `woclr` / `woset` / `rclr` / `rset` shorthand forms.

# Appendix B: Referenced Documents

This appendix collects the companion guides, specifications and external design documentation referenced by the subsystem and IP chapters. Revisions below are those cited in this manual; an unspecified revision does not imply the latest edition.

## B.1. OCAH Documentation

Table 2549 identifies the companion guides used for integration and programming details.

*Table 2549. OCAH documentation references*

| Document | Referenced for |
|---|---|
| Open Chiplet Atlas Harness (OCAH) Integrator Guide | Subsystem integration and external interfaces, including the SEP external TRNG aperture's register decode and error-response requirements. |

## B.2. OCA Specifications

Table 2550 identifies the OCA specifications cited by this manual.

*Table 2550. OCA specification references*

| Document | Referenced for |
|---|---|
| Open Chiplet Cross Triggering (OCCT) specification | DTP cross-trigger fabric, Cross-Trigger Network and Cross-Trigger Port. The cited revision and a document-specific URL are not specified in this manual. |
| Open Chiplet Atlas specification: Open Chiplet Time Synchronization (OCTS) | System Timer OCTS protocol and credit-based synchronization. The cited revision is not specified in this manual. |
| AXI-over-UCIe (AoU) specification | AoU bridge protocol. The imported AoU microarchitecture specification cites revision 0.7 (2025). |

## B.3. Interface and Security Standards

Table 2551 identifies the standards used by the interfaces and security functions.

*Table 2551. Interface and security standard references*

| Document | Referenced for |
|---|---|
| Universal Chiplet Interconnect Express (UCIe) specification | AoU FDI interface, flit cancellation and stall handling. The cited revision is not specified in this manual. |
| IEEE 1149.1-2013 | DTP JTAG TAPs, instruction registers and IC_RESET test data register. |
| IEEE 1838-2019 | DTP hierarchical multi-chiplet test access and STAP 3D control register. |
| IEEE 1687-2014 | DTP iJTAG instrumentation and segment insertion bits (SIBs). |
| I2C-bus specification, revisions 6 and 7.0 | I2C controller. The feature overview cites revision 7.0; the architecture chapter cites revision 6 for electrical and timing requirements. |
| System Management Bus (SMBus) specification, revision 3.2 | I2C controller SMBus operation. |
| Power Management Bus (PMBus) specification, revision 1.4 | I2C controller PMBus operation. |
| AVSBus specification, version 1.3.1 | AVSBus controller voltage-scaling frames and responses. |
| MIPI I3C Basic specification, versions 1.0 and 1.1.1 | I3C core controller interface. |

| Document | Referenced for |
|---|---|
| MIPI I3C HCI (Host Controller Interface) specification, version 1.2 | I3C core controller register map. |
| TCRI recovery specification, version 1.0 | I3C core controller recovery interface. |
| NIST SP 800-90A, revision 1 | DRBG and SEP random-number generation. |
| NIST SP 800-90B | Entropy Source and SEP TRNG entropy assessment and health tests. |
| FIPS 140-2 | Entropy Source discussion of planned certification readiness. |
| FIPS 203 | Adams Bridge ML-KEM-1024 operations. |
| FIPS 204 | Adams Bridge ML-DSA-87 operations and SEP lifecycle signing policy. |

# B.4. External Design Documentation

Table 2552 identifies the upstream design documentation used by the subsystem chapters.

*Table 2552. External design documentation references*

| Document or project | Referenced for |
|---|---|
| Rocket Core | SMC CPU and memory system. |
| VeeR EL2 Programmer's Reference Manual | SEP CPU architecture, CSRs and interrupt behavior. |
| iDMA | SMC DMA engine. |
| Caliptra Adams Bridge documentation | SEP post-quantum cryptographic accelerator. |
| OpenTitan HMAC documentation | SEP HMAC-SHA2 accelerator and programming model. |
| OpenTitan OTBN documentation | SEP public-key accelerator and programming model. |
| OpenTitan AES documentation | SEP AES cipher and programming model. |
| OpenTitan KMAC documentation | SEP KMAC-SHA3 accelerator and programming model. |
| OpenTitan Key Manager | Background on a comparable key manager design. The OCAH Key Manager firmware interface is defined by its own firmware chapter, not by this document. |
| OpenTitan CSRNG and EDN | DRBG core register semantics and application entropy delivery. |
| OpenTitan DMA Controller | SEP DMA and its programming model. |
| OpenTitan SPI Host documentation | SEP SPI host and programming model. |
| OpenTitan Always-On Timer documentation | SEP watchdog timer and programming model. |
| OpenTitan I2C Controller documentation | SMC I2C controller architecture and programming model. The I2C chapter also cites the upstream source. |
| OpenTitan I2C Device Interface Functions | I2C controller firmware interface. |
| OpenTitan Cryptolib API | SEP cryptographic software interface background. |