

# **Security Reference Manual for i.MX 6Dual, 6Quad, 6Solo, and 6DualLite Families of Applications Processors**

Document Number: IMX6DQ6SDLSRM  
Rev. D, 11/2012





## Contents

Section number	Title	Page
<b>Chapter 1 Security Overview</b>		
1.1	Chapter overview.....	19
1.2	Feature summary.....	19
1.3	TrustZone architecture.....	22
1.4	High Assurance Boot (HAB).....	25
1.4.1	HAB process flow.....	25
1.4.2	HAB feature summary.....	27
1.5	Secure Non-volatile Storage Module (SNVS).....	27
1.5.1	SNVS architecture.....	28
1.6	Cryptographic Acceleration And Assurance Module (CAAM).....	29
1.7	OCOTP_CTRL.....	29
1.8	Central Security Unit (CSU).....	30
1.9	AHB to IP Peripheral Bridge (AIPSTZ).....	31
1.10	Digital Transmission Content Protection (DTCP).....	31
1.11	High-bandwidth Digital Content Protection (HDCP).....	32
1.12	System JTAG Controller (SJC).....	33
1.12.1	Scan protection.....	33
1.13	TrustZone Address Space Controller (TZASC-PL380).....	34
1.14	Smart Direct Memory Access Controller (SDMA).....	35
1.15	TrustZone Watchdog (TZ WDOG).....	36
<b>Chapter 2 Security System Integration</b>		
2.1	Master ID allocation.....	37
2.2	System-level SNVS connections.....	37
2.2.1	Unsupported SNVS functions.....	37
2.2.2	SNVS clock tamper input.....	37
2.2.3	System security violation alarm signals monitored by SNVS.....	38

<b>Section number</b>	<b>Title</b>	<b>Page</b>
2.3	Security access error.....	38
2.4	OCRAM TrustZone support.....	39
2.5	WatchDOG mechanism.....	40
2.6	Security configuration.....	40
2.7	Field return for retest procedure.....	41

## **Chapter 3 Central Security Unit (CSU)**

3.1	Overview.....	45
3.1.1	Features.....	45
3.2	Functional description.....	45
3.2.1	Peripheral access policy.....	46
3.2.2	Initialization policy.....	47
3.3	Programmable Registers.....	47
3.3.1	Config security level register (CSU_CSLn).....	49
3.3.2	HP0 register (CSU_HP0).....	54
3.3.3	HP1 register (CSU_HP1).....	58
3.3.4	Secure access register (CSU_SA).....	58
3.3.5	HPCONTROL0 register (CSU_HPCONTROL0).....	62
3.3.6	HPCONTROL1 register (CSU_HPCONTROL1).....	66

## **Chapter 4 System Boot**

4.1	Overview.....	67
4.2	Boot modes.....	68
4.2.1	Boot mode pin settings.....	69
4.2.2	High level boot sequence.....	69
4.2.3	Boot From Fuses Mode (BOOT_MODE[1:0] = 00b).....	70
4.2.4	Serial Downloader.....	71
4.2.5	Internal Boot Mode (BOOT_MODE[1:0] = 0b10).....	72
4.2.6	Boot security settings.....	73

<b>Section number</b>	<b>Title</b>	<b>Page</b>
4.3	Device Configuration.....	74
4.3.1	Boot eFUSE Descriptions.....	74
4.3.2	GPIO Boot Overrides.....	76
4.3.3	Device Configuration Data.....	78
4.4	Device Initialization.....	78
4.4.1	Internal ROM /RAM memory map.....	78
4.4.2	Boot Block Activation .....	80
4.4.3	Clocks at Boot Time.....	81
4.4.4	Enabling MMU and Caches.....	82
4.4.5	Exception Handling.....	83
4.4.6	Interrupt Handling During Boot.....	84
4.4.7	Persistent Bits.....	84
4.5	Boot Devices (Internal Boot).....	85
4.5.1	NOR Flash/OneNAND using EIM Interface.....	85
4.5.1.1	NOR Flash Boot Operation.....	86
4.5.1.2	OneNAND Flash Boot Operation.....	86
4.5.1.3	IOMUX Configuration for EIM Devices.....	87
4.5.2	NAND Flash.....	88
4.5.2.1	NAND eFUSE Configuration.....	89
4.5.2.2	NAND Flash Boot Flow and Boot Control Blocks (BCB).....	90
4.5.2.3	Firmware Configuration Block.....	93
4.5.2.4	Discovered Bad Block Table.....	95
4.5.2.5	Bad Block Handling in the ROM.....	96
4.5.2.6	Toggle Mode DDR NAND Boot.....	97
4.5.2.6.1	GPMI and BCH Clocks Configuration.....	97
4.5.2.6.2	Setup DMA for DDR Transfers.....	98
4.5.2.6.3	Reconfigure Timing and Speed Using Values in FCB.....	98
4.5.2.7	Typical NAND Page Organization.....	98
4.5.2.7.1	BCH ECC Page Organization.....	99

<b>Section number</b>	<b>Title</b>	<b>Page</b>
4.5.2.7.2	Metadata.....	100
4.5.2.8	IOMUX Configuration for NAND.....	100
4.5.3	Expansion Device.....	101
4.5.3.1	Expansion Device eFUSE Configuration.....	101
4.5.3.2	MMC and eMMC Boot.....	104
4.5.3.3	SD, eSD and SDXC.....	112
4.5.3.4	IOMUX Configuration for SD/MMC.....	112
4.5.3.5	Redundant Boot Support for Expansion Device.....	113
4.5.4	Hard Disk and SSD.....	114
4.5.4.1	Hard Disk and SSD eFUSE Configuration.....	115
4.5.4.2	IOMUX and Timing Configuration for SATA.....	115
4.5.4.3	Redundant Boot Support for Hard Disk and SSD.....	116
4.5.5	Serial ROM through SPI and I2C.....	117
4.5.5.1	Serial ROM eFUSE Configuration.....	118
4.5.5.2	I2C Boot.....	119
4.5.5.2.1	I2C IOMUX Pin Configuration.....	120
4.5.5.3	ECSPI Boot.....	120
4.5.5.3.1	ECSPI IOMUX Pin Configuration.....	122
4.6	Program image.....	123
4.6.1	Image Vector Table and Boot Data.....	123
4.6.1.1	Image Vector Table Structure.....	124
4.6.1.2	Boot Data Structure.....	125
4.6.2	Device Configuration Data (DCD).....	125
4.6.2.1	Write Data Command.....	126
4.6.2.2	Check Data Command.....	128
4.6.2.3	NOP Command.....	129
4.6.2.4	Unlock Command.....	130
4.7	Plugin Image.....	130

Section number	Title	Page
4.8	Serial Downloader.....	131
4.8.1	USB.....	132
4.8.1.1	USB Configuration Details.....	133
4.8.1.2	IOMUX Configuration for USB.....	133
4.8.2	Serial Download protocol.....	134
4.8.2.1	SDP Command.....	134
4.8.2.1.1	READ REGISTER.....	135
4.8.2.1.2	WRITE REGISTER.....	135
4.8.2.1.3	WRITE_FILE.....	136
4.8.2.1.4	ERROR_STATUS.....	137
4.8.2.1.5	DCD WRITE.....	138
4.8.2.1.6	JUMP ADDRESS.....	139
4.9	Recovery Devices.....	140
4.10	USB Low Power Boot.....	140
4.11	High Assurance Boot (HAB).....	142
4.11.1	ROM Vector Table Addresses.....	143

## Chapter 5 Cryptographic Acceleration and Assurance Module (CAAM)

5.1	Overview of CAAM ( cryptographic acceleration and assurance module) functionality.....	145
5.2	Feature summary.....	146
5.3	CAAM implementation.....	148
5.3.1	CAAM submodules.....	149
5.3.2	Cryptographic engines implemented in CAAM.....	149
5.4	CAAM modes of operation.....	149
5.4.1	Security Monitor security states.....	150
5.4.1.1	The effect of security state on volatile keys.....	150
5.4.1.2	The effect of security state on non-volatile keys.....	151
5.4.2	Keys available in different security modes.....	151
5.4.2.1	Keys available in trusted mode.....	152

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.4.2.2	Keys available in secure mode.....	152
5.4.2.3	Keys available in non-secure mode.....	153
5.4.2.4	Keys available in fail mode.....	153
5.5	CAAM hardware functional description.....	154
5.5.1	Buses.....	155
5.5.1.1	DMA interface (AXI master bus).....	155
5.5.1.1.1	DMA bursts that may read past the end of data structures.....	155
5.5.1.2	Secure memory interface (AXI slave bus).....	156
5.5.1.3	Register interface (IP bus).....	156
5.5.2	Job execution interfaces.....	157
5.5.2.1	Job ring interface.....	158
5.5.2.1.1	Configuring and managing the input/output rings overview.....	158
5.5.2.1.2	Managing the input rings.....	160
5.5.2.1.3	Managing the output rings.....	161
5.5.2.1.4	Controlling access (rings).....	162
5.5.2.1.5	Order of job completion.....	162
5.5.2.1.6	Initializing job rings.....	162
5.5.2.1.7	Asserting job ring interrupts.....	162
5.5.2.2	Executing job descriptors in single-step mode.....	163
5.5.3	Job scheduling.....	164
5.5.4	Job termination status/error codes.....	166
5.5.5	Job execution hardware.....	169
5.5.5.1	Descriptor controller (DECO) and cryptographic control block (CCB).....	169
5.5.5.1.1	Alignment blocks.....	170
5.5.5.2	Cryptographic hardware accelerators (CHAs) (overview).....	171
5.6	Descriptors and descriptor commands.....	172
5.6.1	Job descriptors.....	172
5.6.2	Trusted descriptors.....	173

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.6.3	Shared descriptors.....	174
5.6.3.1	Executing shared descriptors in proper order.....	175
5.6.3.2	Specifying different types of shared descriptor sharing.....	175
5.6.3.2.1	Error sharing.....	176
5.6.3.3	Changing shared descriptors.....	177
5.6.4	Using in-line descriptors .....	177
5.6.5	Using replacement job descriptors.....	178
5.6.6	Scatter/gather tables (SGTs).....	179
5.6.7	Using descriptor commands.....	180
5.6.7.1	Command execution order.....	180
5.6.7.1.1	Executing commands when SHR = 0.....	181
5.6.7.1.2	Executing commands when SHR = 1.....	182
5.6.7.1.3	Executing commands when REO = 0.....	183
5.6.7.1.4	Executing commands when REO = 1.....	184
5.6.7.1.5	Executing additional HEADER commands.....	184
5.6.7.1.6	Jumping to another job descriptor.....	185
5.6.7.2	Command properties.....	186
5.6.7.2.1	Blocking commands.....	186
5.6.7.2.2	Load/store checkpoint.....	186
5.6.7.2.3	Done checkpoint.....	186
5.6.7.3	Command types.....	186
5.6.7.4	SEQ vs non-SEQ commands.....	188
5.6.7.4.1	Creating a sequence.....	188
5.6.7.4.2	Using sequences for fixed and variable length data.....	189
5.6.7.4.3	Transferring meta data.....	190
5.6.7.4.4	Rewinding a sequence.....	191
5.6.7.5	Information FIFO entries.....	191
5.6.7.6	Cryptographic class.....	191
5.6.7.7	Address pointers.....	192

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.6.8	HEADER command.....	193
5.6.9	KEY commands.....	196
5.6.10	LOAD commands.....	199
5.6.11	FIFO LOAD command.....	208
	5.6.11.1 Bit length data.....	210
	5.6.11.2 FIFO LOAD input data type .....	211
5.6.12	STORE command.....	212
5.6.13	FIFO STORE command.....	217
5.6.14	MOVE and MOVE_LEN commands.....	221
5.6.15	ALGORITHM OPERATION command.....	228
5.6.16	PROTOCOL OPERATION commands.....	232
5.6.17	SIGNATURE command.....	235
5.6.18	JUMP (HALT) command .....	237
	5.6.18.1 Jump type.....	238
	5.6.18.1.1 Local conditional jump.....	238
	5.6.18.1.2 Non-local conditional jump.....	238
	5.6.18.1.3 Conditional halt.....	239
	5.6.18.1.4 Conditional halt with user-specified status.....	239
	5.6.18.1.5 Conditional subroutine call.....	239
	5.6.18.1.6 Conditional subroutine return.....	240
	5.6.18.2 Test type.....	240
	5.6.18.3 JSL and TEST CONDITION fields.....	241
	5.6.18.4 JUMP command format.....	242
5.6.19	MATH command .....	244
5.6.20	SEQ IN PTR command.....	250
5.6.21	SEQ OUT PTR command.....	252
5.7	Cryptographic hardware accelerators (CHAs).....	254
5.7.1	ARC-4 hardware accelerator (AFHA) CHA functionality.....	255
	5.7.1.1 AFHA use of the Mode Register.....	255

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.7.1.2	AFHA use of the Context Register.....	256
5.7.1.3	AFHA use of the Key Register.....	256
5.7.1.4	AFHA use of the Data Size Register.....	257
5.7.1.5	Save and restore operations in AFHA Sbox and AFHA context data.....	257
5.7.1.5.1	What is the Sbox?.....	257
5.7.1.5.2	What are the I and J context pointers?.....	257
5.7.1.5.3	Sbox and context data operations.....	257
5.7.1.6	ARC-4 operation considerations.....	258
5.7.2	Data encryption standard accelerator (DES) functionality.....	259
5.7.2.1	DESA use of the Mode Register.....	259
5.7.2.2	DESA use of the Key Register.....	259
5.7.2.3	DESA use of the Key Size Register.....	260
5.7.2.4	DESA use of the Data Size Register.....	260
5.7.2.5	DESA Context Register.....	260
5.7.2.6	Save and store operations in DESA context data.....	261
5.7.3	Random-number generator (RNG) functionality.....	261
5.7.3.1	RNG features summary.....	261
5.7.3.2	RNG functional description.....	262
5.7.3.2.1	RNG state handles.....	262
5.7.3.2.2	RNG NIST certification.....	262
5.7.3.3	RNG operations.....	263
5.7.3.4	RNG use of the Key Registers.....	264
5.7.3.5	RNG use of the Context Register.....	265
5.7.3.6	RNG use of the Data Size Register.....	265
5.7.4	Message digest hardware accelerator (MDHA) functionality.....	265
5.7.4.1	MDHA use of the Mode Register.....	266
5.7.4.2	MDHA use of the Key Register.....	266
5.7.4.2.1	Using the MDHA Key Register with normal keys.....	267

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.7.4.2.2	Using the MDHA Key Register with IPAD/OPAD "split keys".....	267
5.7.4.2.2.1	Definition and function of IPAD/OPAD split keys.....	267
5.7.4.2.2.2	Process flow of using the Key Register with split keys.....	267
5.7.4.2.2.3	Using padding with the split key type to align with storage.....	268
5.7.4.2.2.4	Length of a split key.....	268
5.7.4.2.2.5	Loading/storing a split key with a key command.....	268
5.7.4.2.2.6	Loading/storing a split key with a FIFO STORE command.....	268
5.7.4.2.2.7	Sizes of split keys.....	269
5.7.4.2.2.8	Constructing an HMAC-SHA-1 split key in memory.....	269
5.7.4.2.3	MDHA use of the Key Size Register.....	270
5.7.4.3	MDHA use of the Data Size Register.....	270
5.7.4.4	MDHA use of the Context Register.....	271
5.7.4.5	Save and restore operations in MDHA context data.....	271
5.7.5	AES accelerator (AES) functionality.....	271
5.7.5.1	What is the AES?.....	271
5.7.5.2	Differences between the encrypt and decrypt keys.....	271
5.7.5.3	AESA modes of operation.....	272
5.7.5.4	AESA use of registers.....	273
5.7.5.5	AESA use of the parity bit.....	273
5.7.5.6	AES ECB mode.....	273
5.7.5.6.1	AES ECB mode use of the Mode Register.....	274
5.7.5.6.2	AES ECB mode use of the Context Register.....	274
5.7.5.6.3	AES ECB Mode use of the Data Size Register .....	275
5.7.5.6.4	AES ECB Mode use of the Key Register.....	275
5.7.5.6.5	AES ECB Mode use of the Key Size Register.....	275
5.7.5.7	AES CBC, OFB, CFB128 modes.....	275
5.7.5.7.1	AES CBC, OFB, and CFB128 modes use of the Mode Register.....	275
5.7.5.7.2	AES CBC, OFB, and CFB128 modes use of the Context Register.....	276
5.7.5.7.3	AES CBC, OFB, and CFB128 modes use of the Data Size Register.....	276

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.7.5.7.4	AES CBC, OFB, and CFB128 modes use of the Key Register.....	277
5.7.5.7.5	AES CBC, OFB, and CFB128 modes use of the Key Size Register.....	277
5.7.5.8	AES CTR mode.....	277
5.7.5.8.1	AES CTR mode use of the Mode Register.....	277
5.7.5.8.2	AES CTR mode use of the Context Register.....	278
5.7.5.8.3	AES CTR mode use of the Data Size Register.....	278
5.7.5.8.4	AES CTR mode use of the Key Register.....	278
5.7.5.8.5	AES CTR mode use of the Key Size Register.....	278
5.7.5.9	AES XCBC-MAC and CMAC modes.....	279
5.7.5.9.1	AES XCBC-MAC and CMAC modes use of the Mode Register.....	279
5.7.5.9.2	AES XCBC-MAC and CMAC Modes use of the Context Register.....	280
5.7.5.9.3	AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register.....	281
5.7.5.9.4	AES XCBC-MAC and CMAC modes use of the Data Size Register.....	281
5.7.5.9.5	AES XCBC-MAC and CMAC modes use of the Key Register.....	281
5.7.5.9.6	AES XCBC-MAC and CMAC modes use of the Key Size Register.....	282
5.7.5.9.7	ICV checking in AES XCBC-MAC and CMAC modes.....	282
5.7.5.10	AESA CCM mode.....	282
5.7.5.10.1	Generation encryption.....	283
5.7.5.10.2	Decryption verification.....	283
5.7.5.10.3	AES CCM mode use of the Mode Register.....	283
5.7.5.10.4	AES CCM mode use of the Context Register.....	284
5.7.5.10.5	AES CCM mode use of the Data Size Register.....	285
5.7.5.10.6	AES CCM mode use of the Key Register.....	286
5.7.5.10.7	AES CCM mode use of the Key Size Register.....	286
5.7.5.10.8	AES CCM mode use of the ICV check.....	286
5.8	Trust Architecture modules.....	286
5.8.1	Secure key module functionality.....	286
5.8.1.1	Initializing and clearing black and trusted descriptor keys.....	287
5.8.1.2	Black keys and JDKEK/TDKEK.....	287

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.8.1.3	Trusted descriptors and TDSK.....	287
5.8.1.4	Master key and blobs.....	288
5.8.2	Black keys.....	288
5.8.2.1	Black key encapsulation schemes.....	288
5.8.2.2	Differences between black and red keys.....	288
5.8.2.3	Loading red keys.....	289
5.8.2.4	Loading black keys.....	289
5.8.2.5	Avoiding errors when loading red and black keys.....	289
5.8.2.6	Encapsulating and decapsulating black keys.....	290
5.8.2.7	Types of black keys and their use.....	291
5.8.2.8	Types of blobs for key storage.....	291
5.8.3	Trusted descriptors.....	292
5.8.3.1	Why trusted descriptors are needed.....	292
5.8.3.2	Trusted-descriptor key types and uses.....	293
5.8.3.3	Trusted descriptors encrypting/decrypting black keys.....	293
5.8.3.4	Trusted-descriptor blob types and uses.....	293
5.8.3.5	Trusted descriptors and secure memory.....	294
5.8.3.6	Configuring the system to create trusted descriptors properly.....	294
5.8.3.7	Creating trusted descriptors.....	295
5.8.3.7.1	Trusted descriptors and descriptor-header bits.....	295
5.8.3.7.2	Trusted-descriptor execution considerations.....	295
5.8.3.7.3	Using master identifier (MID) to access other memory areas .....	296
5.8.4	Blobs.....	296
5.8.4.1	Blob protocol.....	296
5.8.4.2	Why blobs are needed.....	297
5.8.4.3	Blob conformance considerations.....	297
5.8.4.4	Encapsulating and decapsulating blobs.....	299
5.8.4.5	Blob types.....	299
5.8.4.5.1	Blob types differentiated by format.....	300

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.8.4.5.2	Blob types differentiated by content.....	301
5.8.4.5.2.1	Red blobs (for general data).....	301
5.8.4.5.2.2	Black blobs (for cryptographic keys).....	302
5.8.4.5.2.3	Enforcing blob content type.....	302
5.8.4.5.3	Blob types differentiated by security state.....	303
5.8.4.5.4	Blob types differentiated by memory type.....	303
5.8.4.5.4.1	General/secure memory blobs and access control.....	303
5.8.4.5.4.2	Differences between general memory and secure memory blobs.....	304
5.8.4.6	Blob encapsulation.....	304
5.8.4.7	Blob decapsulation.....	305
5.8.4.7.1	Operations.....	306
5.8.5	Critical security parameters.....	307
5.8.6	Secure memory.....	307
5.8.6.1	CAAM secure memory features.....	308
5.8.6.2	Secure memory controller (SMC) states.....	308
5.8.6.2.1	SMC initialize state.....	309
5.8.6.2.2	SMC normal state.....	309
5.8.6.2.3	SMC fail state.....	310
5.8.6.3	Secure memory organization.....	311
5.8.6.4	Secure memory security functions.....	312
5.8.6.4.1	Automatic RAM zeroization.....	312
5.8.6.4.1.1	Zeroizing secure memory marked "CSP".....	313
5.8.6.4.2	Access control.....	313
5.8.6.4.2.1	Access control in secure memory RAM partitions.....	313
5.8.6.4.2.2	Access control through job rings.....	314
5.8.6.4.2.3	Setting access control permissions.....	314
5.8.6.4.3	Cryptographic protection of exported data.....	315
5.8.6.4.3.1	Exporting/importing memory type blobs.....	316
5.8.6.4.3.2	Access permissions cryptographically bound to secure memory blobs...316	

<b>Section number</b>	<b>Title</b>	<b>Page</b>
5.8.6.5	Initializing secure memory.....	316
<b>Chapter 6 Secure Non-Volatile Storage (SNVS)</b>		
6.1	SNVS overview.....	319
6.1.1	SNVS feature summary.....	319
6.2	SNVS structure.....	322
6.3	SNVS _HP (high power part).....	323
6.3.1	System security monitor (SSM).....	324
6.3.1.1	Transitioning among system security monitor states.....	324
6.3.1.2	HP security violation policy.....	327
6.3.1.3	SNVS _HP violation sources.....	327
6.3.2	Master key control.....	328
6.3.2.1	Error Code for the OTPMK.....	329
6.3.2.2	Generating the code bits.....	330
6.3.2.3	Checking the code bits.....	331
6.3.2.4	Error code for the ZMK.....	331
6.3.3	ZMK hardware programming mechanism.....	333
6.3.4	Non-secure real time counter.....	333
6.3.4.1	Calibrating the time counter.....	333
6.3.4.2	Time counter alarm.....	334
6.3.4.3	Periodic interrupt.....	334
6.4	Low power part (SNVS _LP).....	335
6.4.1	Behavior during system power down.....	335
6.4.2	Zeroizable master key (ZMK).....	335
6.4.3	Secure real time counter (SRTC).....	335
6.4.3.1	Calibrating the SRTC time counter.....	336
6.4.3.2	Time counter alarm (zmk).....	336
6.4.4	Monotonic counter (MC).....	336
6.4.5	Power glitch detector (PGD).....	337

<b>Section number</b>	<b>Title</b>	<b>Page</b>
6.4.6	General-Purpose Register.....	338
6.4.7	LP security violation/tamper policy .....	338
6.5	Modes of operation.....	339
6.5.1	Operational states.....	340
6.6	SNVS clock sources.....	340
6.7	SNVS reset and system power up.....	341
6.8	SNVS interrupts, alarms, and security violations.....	341
6.9	Programming Guidelines.....	342
6.9.1	RTC/SRTC control bits setting.....	342
6.9.2	RTC/SRTC value read.....	343
6.9.3	ZMK programming guidelines.....	344
6.9.4	General initialization guidelines.....	345
6.10	SNVS Memory Map/Register Definition.....	345
6.10.1	SNVS_HP Lock Register (SNVS_HPLR).....	348
6.10.2	SNVS_HP Command Register (SNVS_HPCOMR).....	351
6.10.3	SNVS_HP Control Register (SNVS_HPCR).....	355
6.10.4	SNVS_HP Security Interrupt Control Register (SNVS_HPSICR).....	357
6.10.5	SNVS_HP Security Violation Control Register (SNVS_HPSVCR).....	359
6.10.6	SNVS_HP Status Register (SNVS_HPSR).....	361
6.10.7	SNVS_HP Security Violation Status Register (SNVS_HPSVSR).....	363
6.10.8	SNVS_HP High Assurance Counter IV Register (SNVS_PHACIVR).....	364
6.10.9	SNVS_HP High Assurance Counter Register (SNVS_PHACR).....	365
6.10.10	SNVS_HP Real Time Counter MSB Register (SNVS_HPRTCMR).....	365
6.10.11	SNVS_HP Real Time Counter LSB Register (SNVS_HPRTCLR).....	366
6.10.12	SNVS_HP Time Alarm MSB Register (SNVS_HPTAMR).....	366
6.10.13	SNVS_HP Time Alarm LSB Register (SNVS_HPTALR).....	367
6.10.14	SNVS_LP Lock Register (SNVS_LPLR).....	368
6.10.15	SNVS_LP Control Register (SNVS_LPCR).....	370
6.10.16	SNVS_LP Master Key Control Register (SNVS_LPMKCR).....	372

<b>Section number</b>	<b>Title</b>	<b>Page</b>
6.10.17	SNVS_LP Security Violation Control Register (SNVS_LPSVCR).....	374
6.10.18	SNVS_LP Tamper Glitch Filters Configuration Register (SNVS_LPTGFCR).....	376
6.10.19	SNVS_LP Tamper Detectors Configuration Register (SNVS_LPTDCR).....	378
6.10.20	SNVS_LP Status Register (SNVS_LPSR).....	381
6.10.21	SNVS_LP Secure Real Time Counter MSB Register (SNVS_LPSRTCMR).....	384
6.10.22	SNVS_LP Secure Real Time Counter LSB Register (SNVS_LPSRTCLR).....	385
6.10.23	SNVS_LP Time Alarm Register (SNVS_LPTAR).....	385
6.10.24	SNVS_LP Secure Monotonic Counter MSB Register (SNVS_LPSMCMR).....	386
6.10.25	SNVS_LP Secure Monotonic Counter LSB Register (SNVS_LPSMCLR).....	386
6.10.26	SNVS_LP Power Glitch Detector Register (SNVS_LPPGDR).....	387
6.10.27	SNVS_LP General Purpose Register (SNVS_LPGPR).....	387
6.10.28	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR $n$ ).....	388
6.10.29	SNVS_HP Version ID Register 1 (SNVS_HPVIDR1).....	389
6.10.30	SNVS_HP Version ID Register 2 (SNVS_HPVIDR2).....	389

# Chapter 1

## Security Overview

### 1.1 Chapter overview

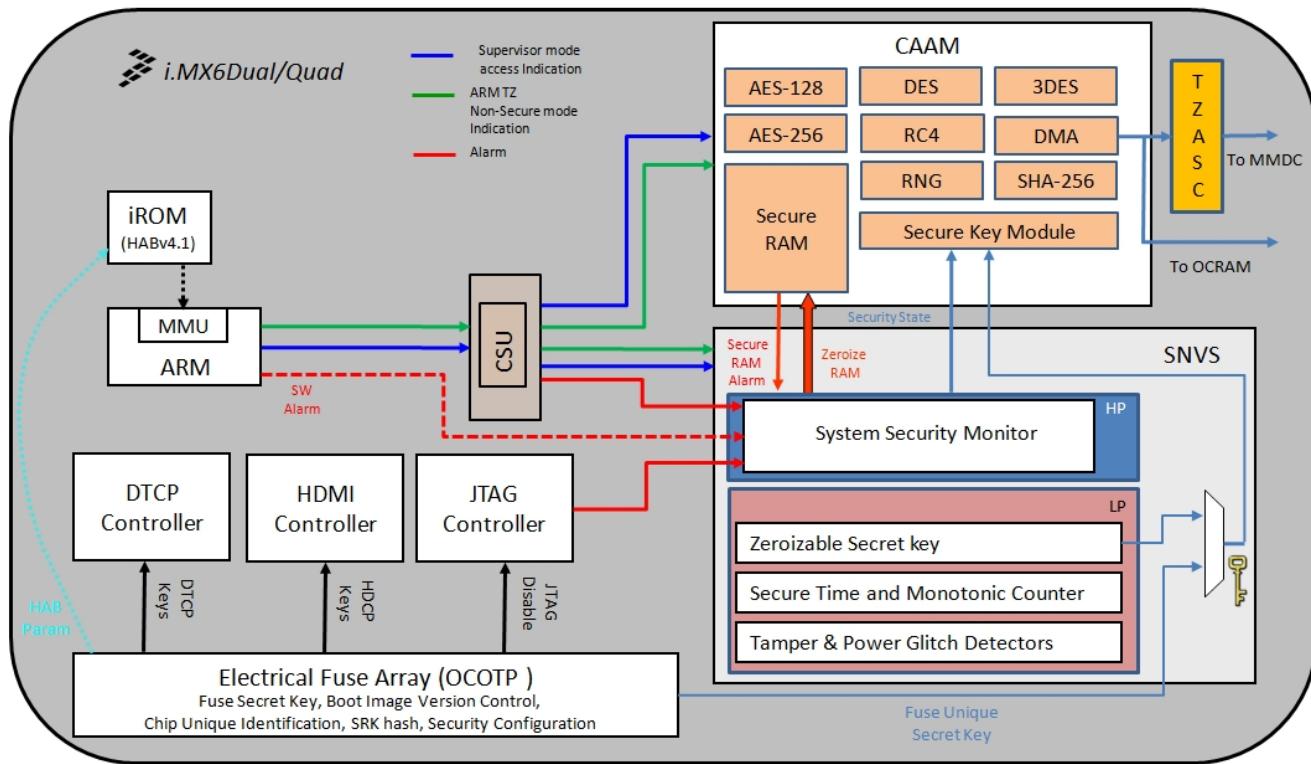
This chapter provides an overview of the following chip security components, explaining the purpose and features of each of them.

- TrustZone (TZ) architecture including: security extensions in the Cortex-A9 processors, Generic Interrupt Controller (GIC), TrustZone Watchdog (TZ WDOG), on-chip RAM (OCRAM) and Trust Zone Address Space Controller (TZASC-PL380)
- High Assurance Boot (HAB) feature in the system boot
- Secure Non-volatile Storage (SNVS) with security monitor, key storage and real-time clock
- Cryptographic Acceleration And Assurance Module (CAAM) with 16 KB of dedicated secure RAM.
- On-chip One-time Programmable Element Controller (OCOTP\_CTRL) with on-chip electrical fuse arrays
- Central Security Unit (CSU)
- Digital Transmission Content Protection (DTCP) controller
- High-bandwidth Digital Content Protection (HDCP) controller
- System JTAG Controller (SJC) with secure debug
- Locked mode in Smart Direct Memory Access Controller (SDMA)

Detailed descriptions of each component are found either in the remaining chapters of this security reference manual or in *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual* (IMX6DQRM) and *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual* (IMX6SDLRM).

### 1.2 Feature summary

The following figure shows a simplified diagram of the security subsystem.



**Figure 1-1. Security subsystem (simplified)**

This diagram represents one example of the use of CSU.

All platforms built using this chip share a general need for security. However, the specific security requirements vary greatly with platform and market. For example, portable consumer devices need to protect a different type and cost of assets than do automotive or industrial platforms. Likewise, each market must protect against different kinds of attacks. Platform designers need an appropriate set of counter measures to meet their specific platform's security needs.

To help platform designers meet the requirements for each market, the chip incorporates a range of security features. Most of these features provide protection against specific kinds of attack, and they can be configured for different levels according to the required degree of protection. These features are designed to work together, and they can be used individually or in concert to underpin the platform security architecture. They can also be integrated with appropriate software to create defensive layers. In addition, the chip includes a general-purpose accelerator that enhances the performance of selected industry-standard cryptographic algorithms.

The security features include:

- ARM TrustZone Architecture, a trusted execution environment for security-critical software

- Hardware-assisted virtualization-two virtual machines: Secure and Normal Worlds (processor modes)
- Hardware firewalls
  - Control access from CPU and DMA peripherals to on-chip peripherals and to both on-chip and off-chip memory
  - Interrupt separation
  - Secure storage separation
  - Cryptographic separation
- Secure High Assurance Boot
  - Security library embedded in tamper-proof on-chip ROM
  - Authenticated boot, which protects against unauthorized software
    - Verification of the code signature during boot
    - RSA-1024/2048/3072/4096 keys anchored to OTP fingerprint (SHA-256)
  - Encrypted boot, which protects software compatibility
  - Runs every time chip is reset
  - Image version control/image revocation (on-chip OTP-based)
- Secure storage
  - On-chip zeroizable secure RAM (16 Kbyte)
  - Off-chip storage protection using AES-256 and chip's unique hardware-only key
- Hardware cryptographic accelerators
  - Symmetric: AES-128, AES-256, DES, 3DES, ARC4
  - Hash message digest and HMAC: SHA-1, SHA-224, SHA-256, MD-5
- True and pseudorandom number generator
- On-chip secure real-time clock with autonomous power domain
- Options for multimedia content link protection
  - HDCP (HDMI connection)
  - DTCP (MOST connection)
  - Supplied device options are: non-adopter, HDCP-adopter or DTCP-adopter, but not both
- Secure debugging
  - Configurable protection against unauthorized JTAG manipulation
  - Three security levels plus complete JTAG disable
  - Support for JTAG port secure reopening for field return debugging
- Universal unique ID
- Electrical fuses (OTP Memory)
- Physical tamper detection
  - Tamper input signal available for cover seal and power glitch detection
  - Hardware and software tamper response

## 1.3 TrustZone architecture

The TrustZone architecture provides a trusted execution environment for security-critical software. Software running in this environment is protected against attacks from potentially compromised platform software, including applications, services, drivers, and even the operating system itself. The TrustZone hardware protects the confidentiality and integrity of both security services and sensitive data. Furthermore, security services cannot be starved of access to processor resources or hijacked by uncontrolled interrupts. TrustZone allows security-critical software to coexist with a rich platform software environment.

The following features work together to create the TrustZone architecture.

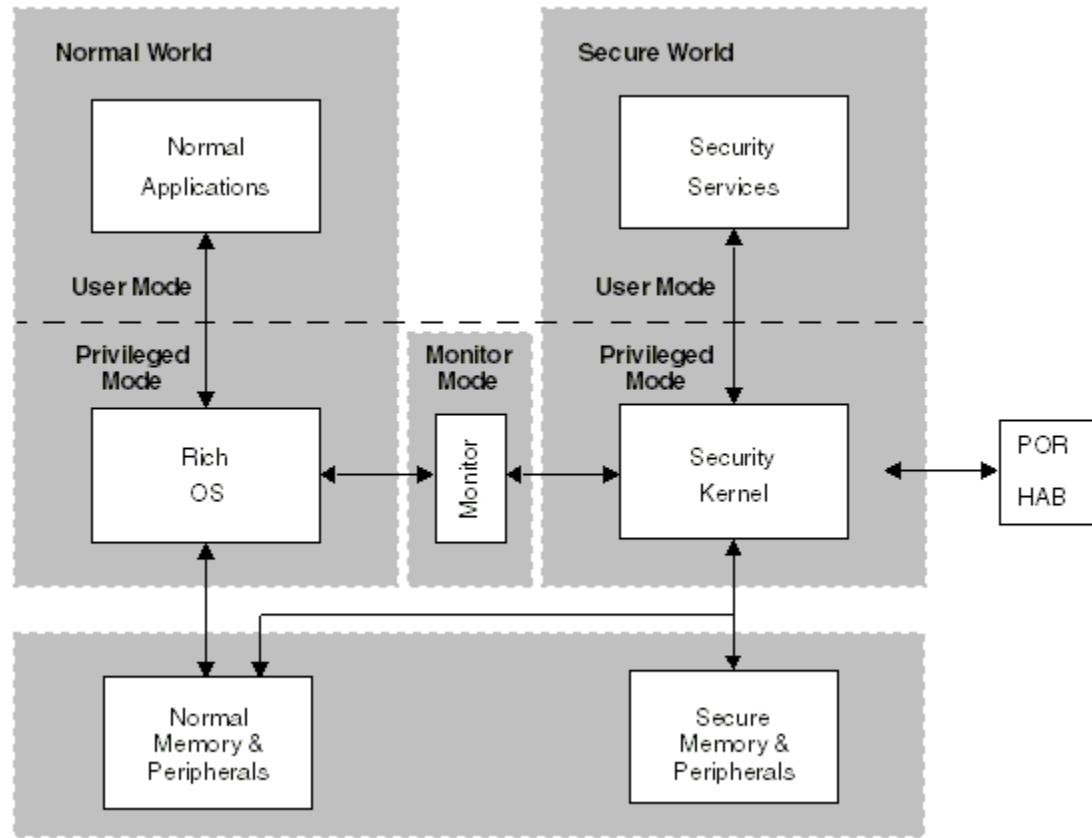
- TrustZone security extensions in the ARM core (see [Figure 1-2](#)) duplicate the user, supervisor and other privileged modes of the processor in a Secure World and a Normal World. Security services execute in Secure World under the control of a security kernel, while normal services and applications run in Normal World with a rich operating system. TrustZone also provides a monitor mode, to which Normal World operating system traps when security services are required.
- TrustZone extensions to the MMU and memory caches separate Secure World and Normal World memory spaces (including memory-mapped peripherals). All read, write and instruction fetch operations from the ARM core indicate the current world, and the page tables and cache lines can be isolated from each other. With this, the security kernel controls access to Secure World memory and peripherals from Normal World software (even the operating system).
- TrustZone Access Controller (TZASC) separates Secure World and Normal World external memory spaces for other bus masters such as DMA-equipped peripherals. More details on TZASC are found in [TrustZone Address Space Controller \(TZASC-PL380\)](#).
- The on-chip RAM (OCRAM) controller separates secure world and normal world internal memory spaces for other bus masters. More details on OCRAM are found in the OCRAM chapter of the *i.MX 6Dual/6Quad Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Applications Processor Reference Manual (IMX6SDLRM)*.
- Central Security Unit (CSU) and the peripheral bridge (AIPS-TZ) separates Secure World and Normal World peripheral address spaces for other bus masters such as DMA-equipped peripherals. More details on CSU are found in [Central Security Unit \(CSU\)](#). Some information on the AIPS-TZ is in this document. Details on AIPS-TZ are found in *i.MX 6Dual/6Quad Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Applications Processor Reference Manual (IMX6SDLRM)*

- Generic Interrupt Controller (GIC) collects interrupt requests from all sources and provides the interrupt interface to the core. Each interrupt source can be configured dynamically as a normal or a secure interrupt by Secure World software. The context switch to handle a normal or secure interrupt when executing in Normal World or Secure World is configured in the ARM core.
- TrustZone Watchdog (TZ WDOG) protects against Normal World software preventing a switch back to Secure World, thereby starving security services of access to the core. Once TZ WDOG is activated, Secure World software must service it on a periodic basis. If servicing does not take place before the configured timeout, TZ WDOG asserts a secure interrupt that forces a switch to Secure World. If it is still not served, TZ WDOG asserts a security violation alarm. TZ WDOG cannot be programmed or deactivated from Normal World. More details on TZ WDOG are found in [TrustZone Watchdog \(TZ WDOG\)](#)

The MMU and the TrustZone architecture are capable of distinguishing between four different code execution modes:

- Code executing in Normal World mode:
  - Code running in kernel mode (also called supervisor mode or privileged mode)
  - Code running in user mode
- Code executing in TrustZone Secure World mode:
  - Code running in TrustZone kernel mode (also called supervisor mode or privileged mode)
  - Code running in TrustZone user mode

The following figure shows the four execution modes.



**Figure 1-2. TrustZone security extensions**

The TrustZone architecture also provides hardware support for a limited virtualization of the ARM core. In the limited virtualization, there are two guest virtual machines: one secure and the other non-secure.

The TrustZone architecture is integrated with other security features for trusted execution support, as follows:

- After power on reset (POR), all ARM cores are in Secure World, all interrupts are secure interrupts, all bus masters are configured as Secure World masters, and all bus slaves can be accessed by Secure World bus masters. This has two implications:
  - If a trusted execution environment is not required, there is no need to switch to Normal World. In this case, the system is backwards compatible with a non-TrustZone system. All platform software runs in Secure World without modification for TrustZone.
  - If enabled, HAB executes in Secure World to authenticate either the security kernel (on a platform using TrustZone) or the normal operating system bootloader.
- The CSU, AIPS, TZASC and CAAM Secure RAM controller enforce configurable core access rights to peripherals and memory from Secure and Normal Worlds.

- The CSU configures other bus masters to make either Secure or Normal World accesses.
- Cryptographic separation on CAAM. The CAAM DMA master makes secure or Normal World accesses according to the configuration set for each job ring.
- Secure storage separation (CAAM/SNVS)
- If not serviced, the TZ WDOG security violation alarm goes to the SNVS.

For more details on the components of the chip TrustZone architecture, see the descriptions of the Cortex A9 Platform in the *i.MX 6Dual/Quad Multimedia Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual (IMX6SDLRM)*.

## 1.4 High Assurance Boot (HAB)

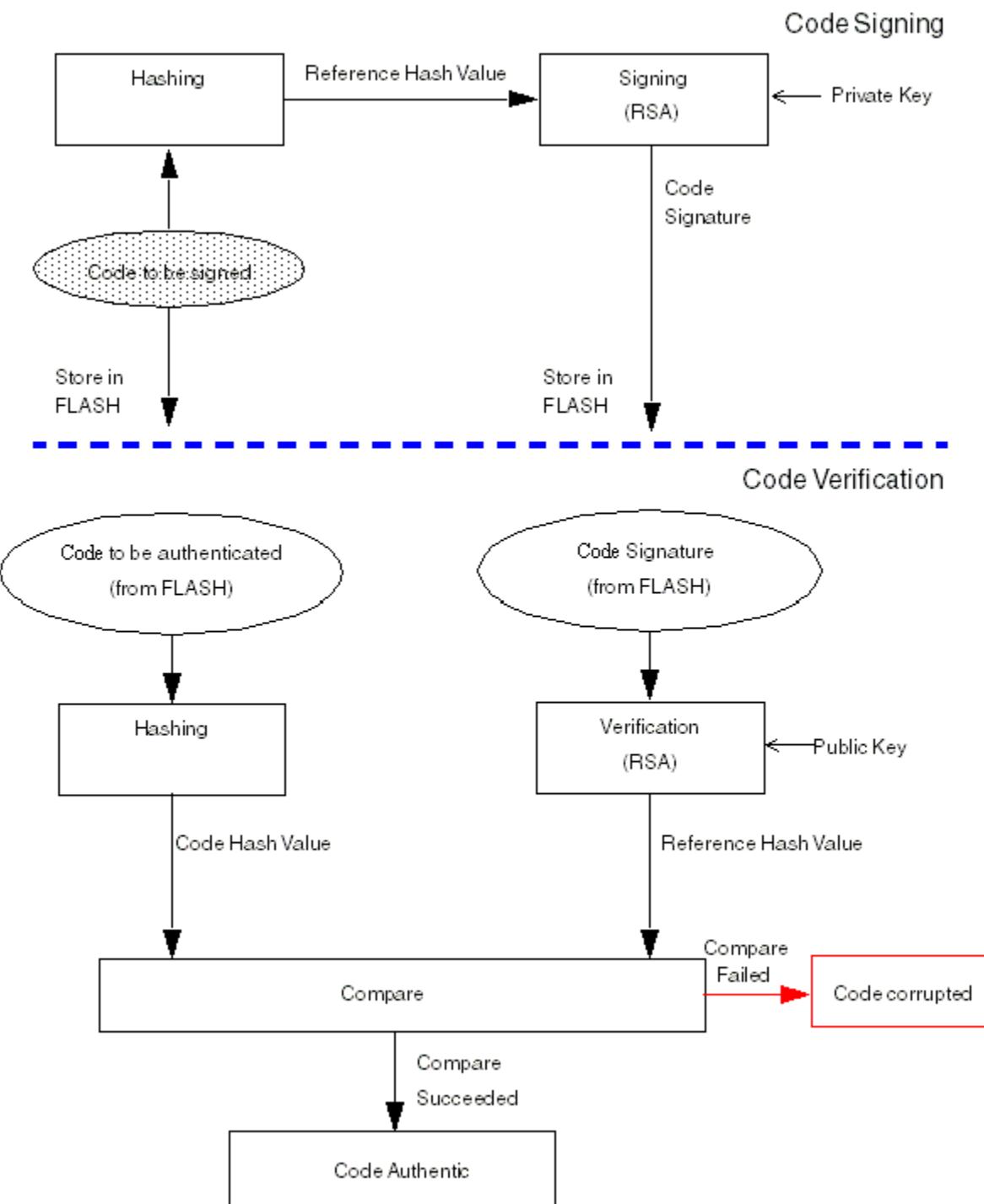
HAB, which is the high assurance boot feature in the system boot ROM, protects the platform from executing unauthorized software (malware) during the boot sequence. When unauthorized software is permitted to gain control of the boot sequence, it can be exploited for a variety of goals, such as exposing stored secrets; circumventing access controls to sensitive data, services, or networks; or repurposing the platform. Unauthorized software can enter the platform during upgrades or reprovisioning, or when booting from USB connections or removable devices.

HAB protects against unauthorized software by:

- Using digital signatures to recognize authentic software. This allows the user to boot the device to a known initial state, running software signed by the device manufacturer.
- Using code encryption to protect the confidentiality of software during off-chip storage. When activated, HAB decrypts the software loaded into RAM prior to execution.

### 1.4.1 HAB process flow

Figure below shows the flow for creating and verifying digital signatures. The top half of this figure shows the signing process, which is performed once off-chip. The bottom half shows the verification process performed on-chip during every system boot.

**Figure 1-3. Code signing and authentication processes**

The original software is programmed in flash memory (or any other boot device) along with the signature. HAB uses a public key to recover the reference hash value from the signature; it then compares the reference hash value to a current hash value produced

from the software in flash. If the contents of the flash have been modified either intentionally or unintentionally, the two hash values do not match and the verification fails.

## 1.4.2 HAB feature summary

HAB incorporates:

- Enforced internal boot via on-chip masked ROM
- Authentication of software loaded from any boot device (including USB download)
- Authenticated decryption of software loaded from any boot device (including USB download) using AES keys (128, 192 or 256 bit)
- CMS PKCS#1 signature verification using RSA public keys (1024 bit to 4096 bit) and the SHA-256 hash algorithm
- Public key infrastructure (PKI) support using X.509v3 certificates
- Root public key fingerprint in manufacturer-programmable on-chip fuses
- Multiple root public keys with revocation by fuses
- Initialization of other security components
- Authenticated USB download failover on any security failure
- Open configuration for development purposes and non-secure platforms
- Closed configuration for shipping secure platforms

On the chip, HAB is integrated with other security features as follows:

- HAB executes in the TrustZone Secure World.
- HAB initializes the SNVS security monitor state machine. Successful secure boot with HAB is required for platform software to gain use of the CAAMDCP master secret key selected by SNVS.
- HAB reads the root public key fingerprint, revocation mask, and security configuration from the OCOTP\_CTRL.
- HAB initializes the CSU.
- HAB can use CAAMDCP to accelerate hash calculations.

See the System Boot chapter for more information on HAB.

## 1.5 Secure Non-volatile Storage Module (SNVS)

Secure Non-volatile Storage Module (SNVS):

- Provides a non-volatile real-time clock maintained by a coin-cell during system power-down for use in both secure and non-secure platforms

## Secure Non-volatile Storage Module (SNVS)

- Protects the real-time clock against rollback attacks in time-sensitive protocols such as DRM and PKI
- Deters replay attacks in time-independent protocols such as certificate or firmware revocation
- Handles tamper detection and tamper reaction to defend sensitive data and operations against compromise, both at run-time and during system power-down
- Controls the access to the OTP master secret key used by CAAM to protect confidential data in off-chip storage
- Provides non-volatile highly-protected storage for an alternative master secret key

### 1.5.1 SNVS architecture

SNVS is partitioned into two sections: a low power part (SNVS\_LP) and a high power part (SNVS\_HP)

The SNVS\_LP block is in the always powered up domain. It is isolated from the rest of the logic by means of isolation cells, which are library-instantiated cells that insure that the powered up logic is not corrupted when power goes down in the rest of the chip.

SNVS\_LP has the following functional units:

- Zeroizable Master Key
- Secure non-rollover real time counter with alarm
- Non-rollover monotonic counter
- Power glitch detector
- Tamper detection monitor
- General-purpose register
- Control and status registers

SNVS\_HP is in the chip power supply domain. SNVS\_HP provides an interface between SNVS\_LP and the rest of the system. Access to SNVS\_LP registers can only be gained through the SNVS\_HP when it is powered up according to access permission policy.

SNVS\_HP has the following functional units:

- IP bus interface
- SNVS\_LP interface
- System Security Monitor (SSM)
- Zeroizable Master Key programming mechanism
- Master Key control block
- Non-secure real time counter with alarm
- Control and status registers

## 1.6 Cryptographic Acceleration And Assurance Module (CAAM)

Cryptographic Acceleration And Assurance Module (CAAM) has two major purposes: acceleration and assurance. It provides:

- Cryptographic acceleration
  - Encryption algorithms: AES, DES/3DES, RC4
  - Hashing algorithms: MD5, SHA-1/224/256
  - Message authentication codes: HMAC, AES-CMAC, AES-XCBC-MAC
  - Authenticated encryption algorithms: AES-CCM
- Secure, hardware random number generation RNG4-Pseudorandom Number Generator (PRNG), certified by National Institute of Standards and Technology (NIST), (Cryptographic Algorithm Validation Program (CAVP), DRBG validation number #94, SHS validation number #1455).
- Export and import of cryptographic blobs
- Secure memory controller and interface
  - Protection for cryptographic keys against exposure to malicious software running on the processor as well as board-level physical attacks
  - Protected storage for confidential data such as proprietary software, encryption keys, passwords or PINs against unauthorized disclosure, including cloning
  - Protected storage for trusted data such as digital rights management (DRM) licences, biometrics reference data, or root certificates against unauthorized modification
  - Both on-chip (run-time) and off-chip protected storage
- Automatic zeroization of security parameters and secure memory
- Two job rings
- IP slave interface
- DMA

## 1.7 OCOTP\_CTRL

OCOTP\_CTRL provides the primary user-visible mechanism for interfacing with on-chip fuses. These fuses' uses include the following:

- Unique chip identifiers
- Mask revision numbers
- Cryptographic keys
- Security configuration

## Central Security Unit (CSU)

- Boot characteristics
- Various control signals requiring permanent non-volatility.

For security purposes, the fuses protect the confidentiality or integrity of critical security data against both software attacks and board-level hardware attacks.

OCOTP\_CTRL provides:

- Shadow cache of fuse values loaded at reset prior to system boot.
- Ability to read and override fuse values in shadow cache (doesn't affect fuse element).
- Ability to read fuses directly (ignoring shadow cache).
- Ability to write (program) fuses by software or JTAG.
- Fuses and shadow cache bits enforcing read-protect, override-protect, and write-protect.
- Lock fuses for selected fuse fields
- Scan protection
- Volatile software-accessible signals which can be used for software control of hardware elements not requiring non-volatility.

## 1.8 Central Security Unit (CSU)

Central Security Unit (CSU) sets access control policies between bus masters and bus slaves, allowing peripherals to be separated into distinct security domains. This protects against indirect unauthorized access to data such as occurs when software programs a DMA bus master to access addresses that the software itself is prohibited from accessing directly. Configuring DMA bus master privileges in CSU consistently with software privileges defends against such indirect unauthorized access.

CSU provides:

- Configuration of peripheral access permissions for those peripherals unable to control their own access permissions
- Configuration of bus master privileges for those bus masters unable to control their own privileges
- TrustZone support to enhance non-TrustZone aware bus masters
- Optional locking of individual CSU settings until the next power-on reset

On the chip, CSU interfaces with:

- ARM TrustZone architecture to assign peripherals to Secure World and Normal World domains

## 1.9 AHB to IP Peripheral Bridge (AIPSTZ)

The AIPSTZ bridge provides programmable access protections for both masters and peripherals. It allows the privilege level of a master to be overridden, forcing it to user-mode privilege, and allows masters to be designated as trusted or untrusted.

Peripherals may require supervisor privilege level for access, may restrict access to a trusted master only, and may be write-protected. IP bus peripherals are subject to access control policies set in both CSU registers and AIPSTZ registers. An access is blocked if it is denied by either policy.

## 1.10 Digital Transmission Content Protection (DTCP)

The Digital Transmission Content Protection (DTCP) co-processor autonomously encrypts and decrypts up to 8 multiple synchronous and isochronous data channels routed through the device. It includes full AKE functionality and an M6 cipher engine for DTCP-MOST. It can also be used together with a separate AES-128 accelerator to enable DTCP-IP. This content protection is required for high quality audio and video content, like DVD, Blu-ray Disc™, and HD-broadcasting services. The DTCP accelerator performs authentication and key exchange and multimedia content cipher processing.

The DTCP features:

- Full AKE and M6 Cipher engine
- DTCP accelerator for sources and sinks
- Independent encryption or decryption of up to 8 channels simultaneously
- Encryption of up to 150 bytes of data per MOST frame at 133 MHz
- Complete AKE math can be accomplished in under 8 million clock cycles; this is equal to about 60 ms at 133 MHz

### NOTE

Devices with nominal part numbers are configured as DTCP-non-adopter devices, with DTCP function permanently disabled. Please contact Freescale Semiconductor representatives for orders of DTCP-adopter (DTCP enabled) devices.

### NOTE

The following device options are available:

- HDCP/DTCP non-adopter

## High-bandwidth Digital Content Protection (HDCP)

- HDCP-adopter
- DTCP-adopter

### NOTE

There is no option for both HDCP-adopter and DTCP adopter devices.

For further information about DTCP specification and functionality, see the the Digital Transmission Licensing Administrator (DTLA) organization's documentation (<http://www.dtcp.com/dtcp.aspx>).

- Volume 1, Revision 1.5.1 (informational version)
- Volume 1, Revision 1.5.1 (Chapters 1-8), available under license from the DTLA
- Volume 2, Revision 1.3 (Chapters 9-10), available under license from the DTLA

## 1.11 High-bandwidth Digital Content Protection (HDCP)

High-bandwidth Digital Copy Protection (HDCP) is a form of digital copy protection that prevents the copying of digital audio and video content as it travels across HDMI connection. The HDCP controller complies with HDCP revision 1.4 specification.

The system prevents HDCP-encrypted content from being played on devices that do not support HDCP. Before sending data, a transmitting device checks that the receiver is authorized to receive it and then the transmitter encrypts the data to prevent eavesdropping as it flows to the receiver. The system also allows for HDCP Repeaters that support downstream HDCP protected interface ports.

### NOTE

Devices with nominal part numbers are configured as HDCP-non-adopter devices, with HDCP function permanently disabled. Please contact Freescale Semiconductor representatives for orders of HDCP-adopter (HDCP enabled) devices.

### NOTE

The available device options are:

- HDCP/DTCP non-adopter
- HDCP-adopter
- DTCP-adopter

### NOTE

There is no option for both HDCP-adopter and DTCP adopter devices.

Contact your local Freescale representative for information on HDCP support.

HDCP ver. 1.4 specification and functionality is described in the Digital Content Protection (DCP) organization's documentation. See *HDCP Specification Rev. 1.4*.

## 1.12 System JTAG Controller (SJC)

The JTAG port provides debug access to hardware blocks, including the ARM processor and the system bus. This allows program control and manipulation as well as visibility to the chip peripherals and memory. The JTAG port must be accessible during initial platform development, manufacturing tests, and general troubleshooting. Given its capabilities, JTAG manipulation is a known attack vector for accessing sensitive data and gaining control over software execution. System JTAG Controller (SJC) protects against the whole range of attacks based on unauthorized JTAG manipulation. It also provides a JTAG port that conforms to IEEE 1149.1 and IEEE 1149.6 (AC) standards for BSR (boundary scan) testing.

SJC provides the following security levels:

- JTAG Disabled-JTAG use is permanently blocked.
- No-Debug-All security sensitive JTAG features are permanently blocked.
- Secure JTAG-JTAG use is restricted (as in the No-Debug level) unless a secret-key challenge/response protocol is successfully executed.
- JTAG Enabled-JTAG use is unrestricted.

Security levels are selected via e-fuse configuration.

### 1.12.1 Scan protection

The chip includes further scan protection logic for SJC modes where JTAG use is allowed. This ensures that access to critical security values is protected as follows.

- The chip is reset upon entering scan mode.
- All modules are reset two clock cycles before receiving a scan enable indication.
- The chip cannot exit scan mode without reset.
- Security modules-including CAAM, SNVS, CSU, and OCOTP\_CTRL-have additional scan protection logic to protect sensitive internal data and functionality.

## TrustZone Address Space Controller (TZASC-PL380)

See the "System JTAG Controller (SJC)" chapter in *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual (IMX6SDLRM)* for more information on the SJC.

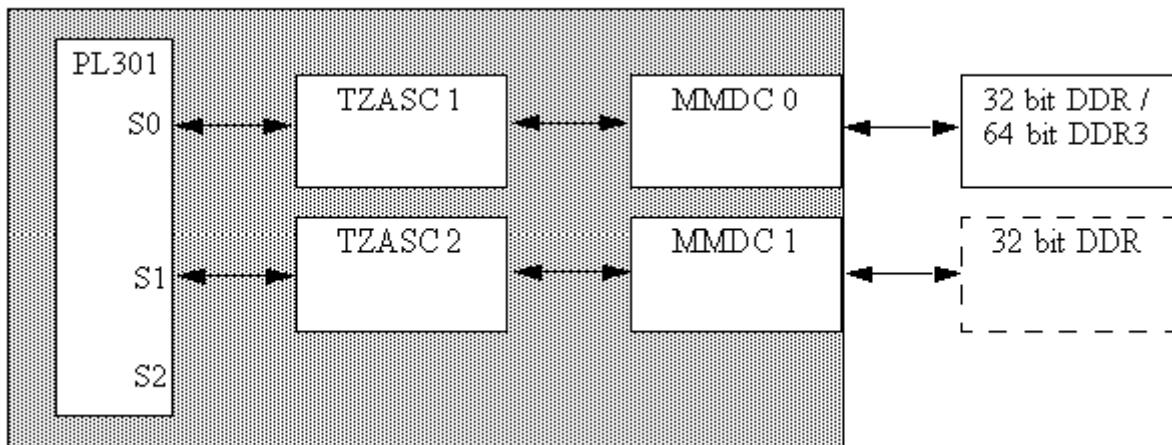
## 1.13 TrustZone Address Space Controller (TZASC-PL380)

TrustZone Address Space Controller (TZASC) protects security-sensitive software and data in a trusted execution environment against potentially compromised software running on the platform.

The TZASC:

- Supports 2, 4, 8, or 16 independent address regions.
- Uses access controls that are independently programmable for each address region and permits data transfers between master and slave only if the security status of the system bus transaction matches the security settings of the memory region it addresses.
- Allows locking of sensitive registers.
- Host interrupt may be programmed to signal attempted access control violations

The chip uses two TZASC instances: one on each of the AXI buses to the two DDR controllers. Each TZASC operates independently of the other. [Figure 1-4](#) shows the TZASC1 and TZASC2 connectivity in the system.



**Figure 1-4. TZASC1 and TZASC2 connectivity in the system.**

The system supports three different external memory maps:

- Two separate 32 bit DDR memory devices. In this case, the memory map allocates 2 Gbyte address space for each memory device. There are two separate data paths: TZASC1n MMDC0 and TZASC2n MMDC1.
- Single 64-bit DDR3 device. In this case, the memory map allocates 4 GByte address space. The data path in this case utilizes only TZASC1n MMDC0.
- Two 32 bit devices in interleaved mode, where the memory map allocates 4 GByte address space on the two devices together, alternating 4 Kbyte block to each device. In this case both TZASC1nMMDC0 and TZASC2nMMDC1 data paths are in use.

The address region access control programming should account for the differences in address mapping schemes. For detailed information about the address mapping in the above cases, refer to the "DDR Mapping to MMDC Controller Port" section in the i.MX 6Dual/Quad and i.MX 6Solo/6DualLite reference manuals (IMX6DQRM, IMX6SDLRM).

See the "TrustZone Address Space Controller (TZASC)" chapter in *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual* (IMX6DQRM) or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual* (IMX6SDLRM) for more information on the TZASC. For a detailed specification of TZASC (PL380) refer to ARM® Infocenter documentation center, Revision: r0p1 *CoreLink TrustZone Address Space Controller TZC-380 Technical Reference Manual*.

## 1.14 Smart Direct Memory Access Controller (SDMA)

Smart Direct Memory Access Controller (SDMA) enables data transfers between peripheral I/O devices and internal/external memories, which maximizes system performance by offloading the CPU in dynamic data routing. Because it is software-programmable, if left unprotected, SDMA could be abused by malicious software to gain indirect access to addresses that the software itself is prohibited from accessing directly. However, SDMA can be configured to protect against this abuse.

SDMA supports two security levels, which are configurable until the next reset by a write-once lock bit:

- In open mode, the processor has full control to load scripts and context into SDMA RAM and modify SDMA registers. This is the default mode.
- In locked mode, selected SDMA registers become read-only to prevent modification of software reset, exception and debug handling. Scripts and their context cannot be loaded into the SDMA RAM anymore.

On the chip, SDMA privileges are configured in the CSU as a further precaution against software abuse.

For more details on the SDMA, refer to the "System DMA (SDMA)" chapter in the *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual (IMX6SDLRM)*.

## **1.15 TrustZone Watchdog (TZ WDOG)**

The TrustZone Watchdog (TZ WDOG) timer module protects against denial of service attacks on Secure World software by Normal World software. TZ starvation is a situation where the normal OS prevents switching to the TZ mode. This situation is undesirable because it can compromise the system's security.

Once the TZ WDOG module is activated, it must be serviced by Secure World software on a periodic basis. If servicing does not take place, the timer expires and the TZ WDOG asserts a secure interrupt that forces a processor switch to Secure World. If this interrupt is not serviced, the TZ WDOG asserts a security violation alarm to the SNVS. The TZ WDOG module cannot be programmed or deactivated by Normal World software.

TZ WDOG is another instantiation of the system WDOG. It enables the following features:

- Time-out periods from 0.5 seconds up to 128 seconds
- Time resolution of 0.5 seconds
- Configurable counters to run or stop during low power modes
- Configurable counters to run or stop during debug mode
- Two event time points: one for TrustZone interrupt assertion and one for security alarm assertion

# Chapter 2

## Security System Integration

### 2.1 Master ID allocation

The following table summarizes the master IDs for all system master modules.

**Table 2-1. Master IDs**

Module	Master ID
Cortex A9	01b
CAAM	10b
SDMA	11b
All others	00b

### 2.2 System-level SNVS connections

#### 2.2.1 Unsupported SNVS functions

The following functions are not supported by the SNVS due to lack of connectivity at the system level:

- Second external tamper input
- Voltage tamper input
- Temperature tamper input
- Two wire-mesh tamper input

## 2.2.2 SNVS clock tamper input

The system provides automatic detection of external SRTC clock state and provides an alternative internal clock source when a failure is detected. When the external clock detection fails, the system switches automatically to an internally generated clock from a ring oscillator on the i.MX6 processor. Once the clock on the RTC\_xtali appears again, the system switches back to the external clock. Thus the clock tamper detection function is reduced to detection of power failure in SNVS power domain or SNVS reset.

## 2.2.3 System security violation alarm signals monitored by SNVS

The SNVS supports six system security violation alarm inputs, as shown in the following table. This allocation is related to HPSVCR and LPSVCR SNVS registers.

**Table 2-2. SNVS system security violation alarm input signals**

SNVS registers			Source	Description
HPSVSR register bit field	HPSVCR register bit field	LPSVCR register bit field		
SEC_VIO0	SV_CFG0	SV_EN0	CAAM	CAAM security violation
SEC_VIO1	SV_CFG1	SV_EN1	SJC	JTAG active
SEC_VIO2	SV_CFG2	SV_EN2	WDOG2	WatchDOG 2 reset
SEC_VIO3	SV_CFG3	SV_EN3	(Reserved)	-
SEC_VIO4	SV_CFG4	SV_EN4	SRC	Internal Boot
SEC_VIO5	SV_CFG5	SV_EN5	IOMUX	External Tamper Detection pad

## 2.3 Security access error

The system slave modules can be configured to return a bus access error when a security-violating access is detected, using the SEC\_ERR\_RESP bit (GPR10[11] register):

- When set, the slave modules return a bus error indication on a non-proper security level access.
- When cleared, the operation does not proceed on a non-proper security level access, but the slave modules do not indicate an error.

The bit is set by default, enabling the error indications. SEC\_RRR\_RESP itself can be locked, preventing further modifications by LOCK\_SEC\_ERR\_RESP bit (GPR10[27]) to assure the system security integrity.

For more information, see the "General-Purpose Registers 10" section in the *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual (IMX6SDLRM)*.

## 2.4 OCRAM TrustZone support

OCRAM supports TrustZone and non-TrustZone accesses to internal, on-chip RAM. There is an option to configure TrustZone only access region.

When OCRAM\_TZSECURE\_REGION[SECURE\_ENBL] bit in the OCRAM module is set, the STARTADDR and ENDADDR bit-fields in this register establish the region of OCRAM that can only be accessed (both read and write) according to the execution mode policy defined in the "Execution Mode Access Policy" section of the CSU chapter. If this bit is cleared, the entire OCRAM can be accessed in either secure or non-secure mode.

The TrustZone bits are described in the "Programmable Registers" section in *i.MX 6Dual/6Quad Multimedia Applications Processor Reference Manual (IMX6DQRM)* or *i.MX 6Solo/6DualLite Multimedia Applications Processor Reference Manual (IMX6SDLRM)*.

### NOTE

The ENDADDR is not configurable. Its value is the last address of the OCRAM space. The STARTADDR granularity is of 4 Kbytes.

The following figure shows OCRAM schematic connectivity.

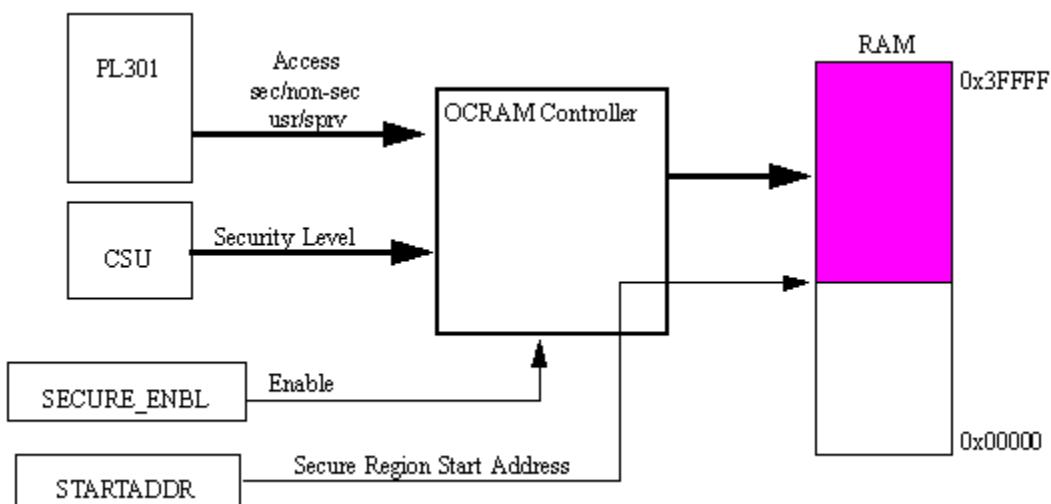


Figure 2-1. OCRAM schematic connectivity

## 2.5 WatchDOG mechanism

The chip has two WDOG modules: WDOG1 and WDOG2 (TZ). Both modules are disabled by default after reset. WDOG1 is configured during the boot whereas WDOG2 is dedicated for secure world purposes and is only activated by TrustZone software if required.

The WDOG module operates as follow:

- If servicing does not take place, the timer times out and asserts the internal system reset signal (wdog\_rst\_B), which goes to SRC, the system reset controller.
- Interrupt can be generated before the counter actually times out.
- wdog\_rst\_B signal can be activated by software.
- There is a power down counter that is enabled out of any reset. This counter has a fixed time out period of 16 seconds after which it asserts the WDOG\_B signal.

## 2.6 Security configuration

Following figure illustrates the typical device security configuration lifecycle, starting from IC fabrication and continuing with OEM development and assembly, through to the final product in the end user's hands. It also shows the option for field return debugging and re-testing at either the OEM facilities or Freescale. Note that Field Return configuration is required for Freescale to run test patterns even on non-secure products (Open configuration), for example if DTCP is activated by burning the DIR\_BT\_DIS fuse.

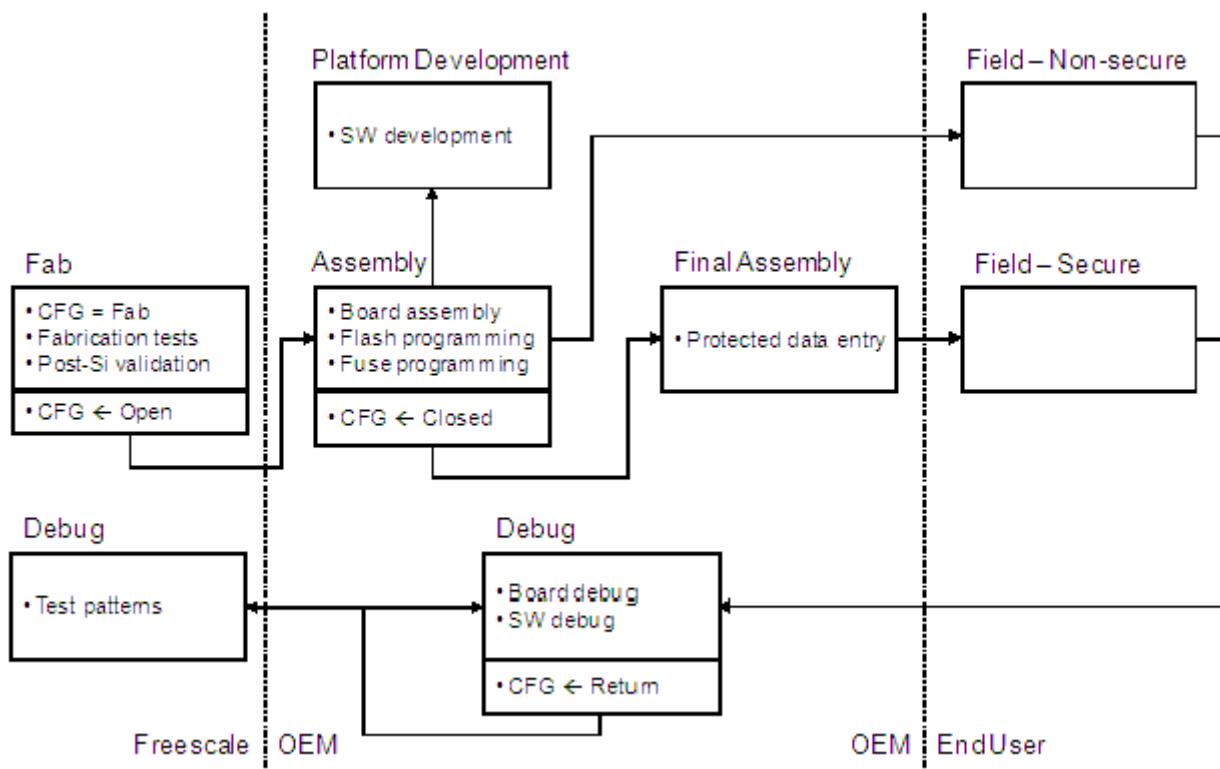


Figure 2-2. Device security configuration life cycle

## 2.7 Field return for retest procedure

On devices shipped to end users, manufacturers can enable debugging restrictions designed to protect device keys and other sensitive data. These debugging restrictions include the following measures.

- Disable Freescale test modes (using the DIR\_BT\_DIS fuse). Note that this measure is required in order to activate DTCP on parts with DTCP enabled.
- Disable JTAG (using JTAG\_SMODE or SJC\_DISABLE fuses).
- Prevent execution of unauthorized bootloader software (using the SEC\_CONFIG[1] fuse).

Naturally, these debugging restrictions also constrain legitimate debugging of field return devices with suspected faults. The chip includes a Field Return configuration to allow legitimate debugging, including the possibility of Freescale running test modes, on returned parts. This Field Return configuration:

- Enables Freescale test modes (overriding the DIR\_BT\_DIS fuse),

#### Field return for retest procedure

- Enables JTAG (overriding JTAG\_SMODE and SJC\_DISABLE fuses),
- Enables execution of unsigned bootloader software as in Open configuration (overriding the SEC\_CONFIG[1] fuse).

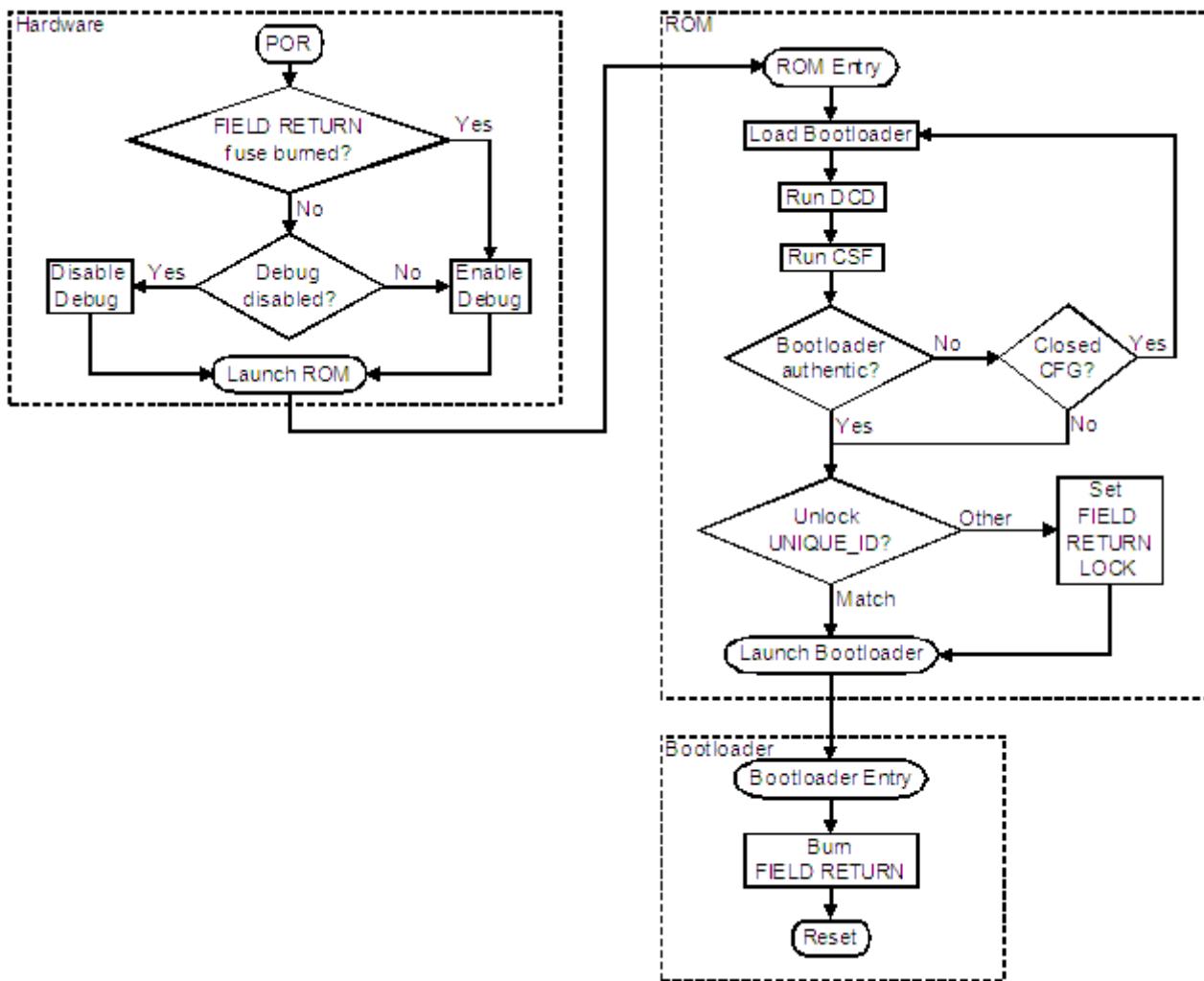
To protect sensitive data already provisioned, the Field Return configuration permanently disables access to device keys (including access from CAAM or DTCP modules).

The entry to the Field Return configuration is strictly controlled in order to deter inadvertent, unauthorized or widespread use.

- The FIELD\_RETURN fuse is protected by the FIELD\_RETURN\_LOCK sticky bit in the OCOTP\_CTRL fuse controller.
- Before leaving the boot ROM, the FIELD\_RETURN\_LOCK bit is set by default so the FIELD\_RETURN fuse cannot be burned.
- Setting the FIELD\_RETURN\_LOCK bit can be avoided by including an Unlock command in the CSF or DCD (Open configuration only) providing:
  - the CSF and bootloader software pass signature verification (Closed configuration), and
  - the Unlock command arguments match the value in the UNIQUE\_ID fuses on the device.

The typical mass production bootloader on shipping devices has no such Unlock command, so entry to Field Return configuration requires booting to a special bootloader which is customized for a single device. For Closed devices, the special bootloader must be signed for that single device, so it cannot be used to unlock other devices even if it is leaked to end users.

The boot flow to activate Field Return configuration is shown in next figure.



**Figure 2-3. HAB FIELD\_RETURN flow chart**

Once FIELD\_RETURN fuse is burned, the part is nearly returned to its OPEN state after the next POR reset, including:

- Revert back value of DIR\_BT\_DIS fuse, to allow Freescale test mode run.
- Enable JTAG (overriding blocking by other means).
- Allow unsigned images to execute, as in OPEN configuration
- Block access to sensitive keys provided by OCOTP\_CTRL or SNVS to DTCP or CAAM
- Note that Field Return configuration is required for Freescale to run test patterns even on non-secure products (Open configuration), for example if DTCP is activated by burning the DIR\_BT\_DIS fuse.



# **Chapter 3**

## **Central Security Unit (CSU)**

### **3.1 Overview**

The CSU manages the system security policy for peripheral access on the SoC. The CSU allows trusted code to set individual security access privileges on each of the peripherals, using one of eight security access privilege levels. Also, according to programmed policy, the CSU may assign bus master security privileges during bus transactions.

#### **3.1.1 Features**

The Central Security Unit (CSU) sets access control policies between bus masters and bus slaves, allowing peripherals to be separated into distinct security domains. This protects against unauthorized access to data e.g. when software programs a DMA bus master to access addresses that the software is prohibited from accessing directly. Configuring DMA bus master privileges in the CSU consistent with software privileges defends against such attempted accesses. Additionally, the CSU manages system security alarms. These alarms are signals routed from various SoC peripherals and I/O that indicate security violation conditions.

CSU has the following security related features:

- Peripheral access policy - Appropriate bus master privilege and identity are required to access each peripheral.
- Masters privilege policy - CSU overrides bus master privilege signals, i.e. user/supervisor secure/non-secure, according to access control policy.

### **3.2 Functional description**

The CSU enables secure software to set bus privilege security policy within the platform.

## Functional description

Security policies may be set, and optionally locked in the CSU registers. Examples of secure software include the command sequence file (CSF) processed by the High Assurance Boot (HAB) or a HAB-authenticated image which executes after the boot ROM.

### 3.2.1 Peripheral access policy

According to its programmed policy, the CSU determines the bus master privileges and the masters that are allowed to access each of the slave peripherals.

There are four security modes of operation (i.e. bus privileges) in the system distinguished by security (TrustZone/non-TrustZone) and privilege (Supervisor/User) setting of the module. Below is the list of these security modes from the highest security level to the lowest:

- TrustZone (Secure) Privilege (Supervisor) Mode - Highest Security Level
- TrustZone (Secure) non-Privilege (User) Mode - Medium Security Level
- non-TrustZone (Regular) Privilege (Supervisor) Mode - Medium Security Level
- non-TrustZone (Regular) non-Privilege (User) Mode - Lowest Security Level

This functionality is implemented as follows:

The Configure Slave Level (CSL) Register value for a specified peripheral resource defines the output signal -- csu\_sec\_level for that peripheral. The value of this signal determines by what master privileges a peripheral is accessible. The relationship between the value of the csu\_sec\_level signal and security operation mode is shown in the table below.

**Table 3-1. Permission Access Table**

CSU_SEC_LEVEL[2:0]	Non-Secure User Mode	Non-Secure Spvr Mode	Secure (TZ) User Mode	Secure (TZ) Spvr Mode	CSL register value
(0) 000	RD+WR	RD+WR	RD+WR	RD+WR	8'b1111_1111
(1) 001	None	RD+WR	RD+WR	RD+WR	8'b1011_1011
(2) 010	RD	RD	RD+WR	RD+WR	8'b0011_1111
(3) 011	None	RD	RD+WR	RD+WR	8'b0011_1011
(4) 100	None	None	RD+WR	RD+WR	8'b0011_0011
(5) 101	None	None	None	RD+WR	8'b0010_0010
(6) 110	None	None	RD	RD	8'b0000_0011
(7) 111	None	None	None	None	Any other value

For more detailed information on the CSU, see the *Multimedia Applications Processor Security Reference Manual for i.MX 6Dual/6Quad and i.MX 6Solo/6DualLite*.

### 3.2.2 Initialization policy

The recommended initialization procedure is as follows:

1. Write the CSU\_CSL Register Field value to indicate each peripheral privilege mode.
2. Write the HP Register Field value to override a Master's privilege mode.

#### NOTE

After programming, to prevent further modification to a register value the register Lock bit should be set.

## 3.3 Programmable Registers

The following sections provide detailed descriptions of the CSU registers and their respective bit and field assignments. Assume that the base address is 021C.

- The CSU registers: CSU\_CSL, CSU\_HP, CSU\_SA and CSU\_HPCONTROL can only be written in Secure Supervisor Mode. (Note: These registers are also referred to as Security Control Registers or SCRs in this document)
- The previous cycle's Lock bit is checked while writing to a register. If the Lock bit was cleared in the previous cycle and is being set during the current cycle, then the register fields covered by that Lock bit may be written during the current cycle.

**CSU memory map**

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
0	Config security level register (CSU_CSL0)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
4	Config security level register (CSU_CSL1)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
8	Config security level register (CSU_CSL2)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
C	Config security level register (CSU_CSL3)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
10	Config security level register (CSU_CSL4)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
14	Config security level register (CSU_CSL5)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
18	Config security level register (CSU_CSL6)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
1C	Config security level register (CSU_CSL7)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
20	Config security level register (CSU_CSL8)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
24	Config security level register (CSU_CSL9)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
28	Config security level register (CSU_CSL10)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
2C	Config security level register (CSU_CSL11)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>

*Table continues on the next page...*

## CSU memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/page
30	Config security level register (CSU_CSL12)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
34	Config security level register (CSU_CSL13)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
38	Config security level register (CSU_CSL14)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
3C	Config security level register (CSU_CSL15)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
40	Config security level register (CSU_CSL16)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
44	Config security level register (CSU_CSL17)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
48	Config security level register (CSU_CSL18)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
4C	Config security level register (CSU_CSL19)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
50	Config security level register (CSU_CSL20)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
54	Config security level register (CSU_CSL21)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
58	Config security level register (CSU_CSL22)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
5C	Config security level register (CSU_CSL23)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
60	Config security level register (CSU_CSL24)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
64	Config security level register (CSU_CSL25)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
68	Config security level register (CSU_CSL26)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
6C	Config security level register (CSU_CSL27)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
70	Config security level register (CSU_CSL28)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
74	Config security level register (CSU_CSL29)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
78	Config security level register (CSU_CSL30)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
7C	Config security level register (CSU_CSL31)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
80	Config security level register (CSU_CSL32)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
84	Config security level register (CSU_CSL33)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
88	Config security level register (CSU_CSL34)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
8C	Config security level register (CSU_CSL35)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
90	Config security level register (CSU_CSL36)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
94	Config security level register (CSU_CSL37)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
98	Config security level register (CSU_CSL38)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
9C	Config security level register (CSU_CSL39)	32	R/W	0033_0033h	<a href="#">3.3.1/49</a>
200	HP0 register (CSU_HP0)	32	R/W	0000_0000h	<a href="#">3.3.2/54</a>
204	HP1 register (CSU_HP1)	32	R/W	0000_0000h	<a href="#">3.3.3/58</a>
218	Secure access register (CSU_SA)	32	R/W	0000_0000h	<a href="#">3.3.4/58</a>
358	HPCONTROL0 register (CSU_HPCONTROL0)	32	R/W	0000_0000h	<a href="#">3.3.5/62</a>
35C	HPCONTROL1 register (CSU_HPCONTROL1)	32	R/W	0000_0000h	<a href="#">3.3.6/66</a>

### 3.3.1 Config security level register (CSU\_CSL $n$ )

There are 40 Config Security Level (CSU\_CSL0-CSU\_CSL39) registers. Each CSU\_CSL is comprised of two fields, each field used to determine the read and write access permissions for a slave peripheral. These 8-bit fields for the first and second slaves are in the locations b23-b16 and bits b7-b0, respectively.

Permission Access Table [Table 3-1](#) shows security levels and csu\_sec\_level signal levels corresponding to different values of the 8-bit CSU\_CSL field for a given slave.

Most slaves have unique CSL registers. Some slaves are grouped together in USB, Timers, PowerUp and Audio groups. The following table shows allocation of CSL register per slave or group of slave modules.

**Table 3-4. CSL Slave Modules Mapping**

Slave Module	Corresponding CSL register and bit field	Comments
PWM1	CSL0 [7:0]	Audio group shared control
PWM2		
PWM3		
PWM4		
CAN1	CSL0 [23:16]	
CAN2	CSL1 [7:0]	
GPT	CSL1 [23:16]	Timers group
EPIT1		
EPIT2		
GPIO1	CSL2 [7:0]	GPIO1 and GPIO2 group
GPIO2		
GPIO3	CSL2 [23:16]	GPIO3 and GPIO4 group
GPIO4		
GPIO5	CSL3 [7:0]	GPIO5 and GPIO6 group
GPIO6		
GPIO7	CSL3 [23:16]	
KPP	CSL4 [7:0]	
WDOG1	CSL4 [23:16]	
WDOG2	CSL5 [7:0]	
CCM		
SNVS_HP	CSL5 [23:16]	Power group
SRC		
GPC		
IP2APB_ANATOP	CSL6 [7:0]	

*Table continues on the next page...*

**Table 3-4. CSL Slave Modules Mapping (continued)**

Slave Module	Corresponding CSL register and bit field	Comments
IOMUXC	CSL6 [23:16]	
DCIC1	CSL7 [7:0]	
DCIC2		
SDMA (port IPS_HOST) EPDC for i.MX6SDL only LCDIF for i.MX6SDL only PXP for i.MX6SDL only	CSL7 [23:16]	
USBOH3 (port PL301) USBOH3 (port USB)	CSL8 [7:0]	
ENET	CSL8 [23:16]	
MLB150	CSL9 [7:0]	
USDHC1	CSL9 [23:16]	
USDHC2	CSL10 [7:0]	
USDHC3	CSL10 [23:16]	
USDHC4	CSL11 [7:0]	
I2C1	CSL11 [23:16]	
I2C2	CSL12 [7:0]	
I2C3	CSL12 [23:16]	
ROMCP	CSL13[7:0]	
VPU	CSL13 [23:16]	MMDC Group
MMDC_CORE (port IPS_P0) MMDC_CORE (port IPS_P1)		
WEIM	CSL14 [7:0]	
OCOTP_CTRL	CSL14 [23:16]	
Reserved	CSL15 [7:0]	
PERFMON1 PERFMON2 PERFMON3	CSL15 [23:16]	PerfMon group
TZASC1	CSL16 [7:0]	
TZASC2	CSL16 [23:16]	
AUDMUX	CSL17 [7:0]	
CAAM	CSL17 [23:16]	
SPDIF	CSL18 [7:0]	
eCSPI1	CSL18 [23:16]	
eCSPI2	CSL19 [7:0]	
eCSPI3	CSL19 [23:16]	
eCSPI4	CSL20 [7:0]	

*Table continues on the next page...*

**Table 3-4. CSL Slave Modules Mapping (continued)**

Slave Module	Corresponding CSL register and bit field	Comments
eCSP15 Reserved for i.MX6SDL	CSL20 [23:16]	
UART1	CSL21 [7:0]	
ESAI1	CSL21 [23:16]	
SSI1	CSL22 [7:0]	
SSI2	CSL22 [23:16]	
SSI3	CSL23 [7:0]	
ASRC (VIA IPSYNC)	CSL23 [23:16]	
Reserved	CSL24 [7:0]	
ROMCP	CSL24 [23:16]	
Reserved	CSL25 [7:0]	
Reserved	CSL25 [23:16]	
Reserved	CSL26 [7:0]	
Reserved	CSL26 [23:16]	
APBH_DMA	CSL27 [7:0]	
HDMI	CSL27 [23:16]	
GPU3D	CSL28[7:0]	
SATA	CSL28 [23:16]	
Reserved for i.MX6SDL		
OPENVG Reserved for i.MX6SDL	CSL29 [7:0]	
ARM core platform DAP and platform controller	CSL29 [23:16]	
HSI	CSL30 [7:0]	
IPU1	CSL30 [23:16]	
IPU2	CSL31 [7:0]	
Reserved for i.MX6SDL		
WEIM	CSL31 [23:16]	
PCIE	CSL32 [7:0]	
GPU2D	CSL32 [23:16]	
MIPI_CORE_CSI	CSL33 [7:0]	
MIPI_CORE_HSI	CSL33 [23:16]	
VDOA	CSL34 [7:0]	
UART2	CSL34 [23:16]	
UART3	CSL35 [7:0]	
UART4	CSL35 [23:16]	
UART5	CSL36 [7:0]	
I2C4 for i.MX6SDL only		
DTCP	CSL36 [23:16]	

*Table continues on the next page...*

**Table 3-4. CSL Slave Modules Mapping (continued)**

Slave Module	Corresponding CSL register and bit field	Comments
Reserved	CSL37 [7:0]	
Reserved	CSL37 [23:16]	
Reserved	CSL38 [7:0]	
Reserved	CSL38 [23:16]	
SPBA	CSL39 [7:0]	
Reserved	CSL39 [23:16]	

**NOTE**

Do not modify the following peripherals' CSL register bits while they are being accessed through the AHB/AXI slave bus:  
EIM, IPU, DTCP, APBHDMA and PCIe.

Address: 0h base + 0h offset + (4d × i), where i=0d to 39d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									NSW_S1	NUW_S1	SSW_S1	SUW_S1	NSR_S1	NUR_S1	SSR_S1	SUR_S1
W									0	0	1	1	0	0	1	1
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									NSW_S2	NUW_S2	SSW_S2	SUW_S2	NSR_S2	NUR_S2	SSR_S2	SUR_S2
W									0	0	1	1	0	0	1	1
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1

**CSU\_CSLn field descriptions**

Field	Description
31–25 -	This field is reserved. Reserved
24 LOCK_S1	Lock bit corresponding to the first slave. Written by secure software. 0 Not locked. Bits 16-23 may be written by software 1 Bits 16-23 locked and cannot be written by software
23 NSW_S1	Non-secure supervisor write access control for the first slave 0 Non-secure supervisor write access disabled for the first slave. 1 Non-secure supervisor write access enabled for the first slave
22 NUW_S1	Non-secure user write access control for the first slave

Table continues on the next page...

**CSU\_CSLn field descriptions (continued)**

<b>Field</b>	<b>Description</b>
	0 Non-secure user write access disabled for the first slave. 1 Non-secure user write access enabled for the first slave.
21 SSW_S1	Secure supervisor write access control for the first slave 0 Secure supervisor write access disabled for the first slave. 1 Secure supervisor write access enabled for the first slave.
20 SUW_S1	Secure user write access control for the first slave 0 Secure user write access disabled for the first slave. 1 Secure user write access enabled for the first slave.
19 NSR_S1	Non-secure supervisor read access control for the first slave 0 Non-secure supervisor read access disabled for the first slave. 1 Non-secure supervisor read access enabled for the first slave.
18 NUR_S1	Non-secure user read access control for the first slave 0 Non-secure user read access disabled for the first slave. 1 Non-secure user read access enabled for the first slave.
17 SSR_S1	Secure supervisor read access control for the first slave 0 Secure supervisor read access disabled for the first slave. 1 Secure supervisor read access enabled for the first slave.
16 SUR_S1	Secure user read access control for the first slave 0 Secure user read access disabled for the first slave. 1 Secure user read access enabled for the first slave
15–9 -	This field is reserved. Reserved
8 LOCK_S2	Lock bit corresponding to the second slave. Written by secure software. 0 Not locked. Bits 7-0 may be written by software 1 Bits 7-0 locked and cannot be written by software
7 NSW_S2	Non-secure supervisor write access control for the second slave 0 Non-secure supervisor write access disabled for the second slave. 1 Non-secure supervisor write access enabled for the second slave
6 NUW_S2	Non-secure user write access control for the second slave 0 Non-secure user write access disabled for the second slave. 1 Non-secure user write access enabled for the second slave.
5 SSW_S2	Secure supervisor write access control for the second slave 0 Secure supervisor write access disabled for the second slave. 1 Secure supervisor write access enabled for the second slave.
4 SUW_S2	Secure user write access control for the second slave 0 Secure user write access disabled for the second slave. 1 Secure user write access enabled for the second slave.

*Table continues on the next page...*

## CSU\_CSLn field descriptions (continued)

Field	Description
3 NSR_S2	Non-secure supervisor read access control for the second slave 0 Non-secure supervisor read access disabled for the second slave. 1 Non-secure supervisor read access enabled for the second slave.
2 NUR_S2	Non-secure user read access control for the second slave 0 Non-secure user read access disabled for the second slave. 1 Non-secure user read access enabled for the second slave.
1 SSR_S2	Secure supervisor read access control for the second slave 0 Secure supervisor read access disabled for the second slave. 1 Secure supervisor read access enabled for the second slave.
0 SUR_S2	Secure user read access control for the second slave 0 Secure user read access disabled for the second slave. 1 Secure user read access enabled for the second slave

## 3.3.2 HP0 register (CSU\_HP0)

The SCU\_HP0 and SCU\_HP1 registers may be programmed to determine the privilege (either User Mode or Supervisor Mode) for seventeen different master groups. The privilege of a particular master group may be overridden by muxing it with the corresponding bit in this register.

The even bit positions (CSU\_HP0[30,28,...0] and CSU\_HP1[0]) in the registers hold the privilege indicator bits; while the odd bit positions (CSU\_HP0[31,29,...,1] and CSU\_HP1[1]) contain lock bits which enable/disable writing to the corresponding privilege indicator bits.

Address: 0h base + 200h offset = 200h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	L_USDHC4	HP_USDHC4	L_USDHC3	HP_USDHC3	L_USDHC2	HP_USDHC2	L_USDHC1	HP_USDHC1	L_DAP	HP_DAP	L_ENET	HP_ENET	L_ABPHDMA	HP_APBHDMA	L_RAWNAND	HP_RAWNAND
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	L_CAAAM	HP_CAAAM	L_MLB	HP_MLB	L_TEST	HP_TEST	L_USB	HP_USB	L_PU	HP_PU	L_SDMA	HP_SDMA	L_SATA	HP_SATA	L_PCIE	HP_PCIE
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**CSU\_HP0 field descriptions**

<b>Field</b>	<b>Description</b>
31 L_USDHC4	Lock bit set by TZ software for HP_USDHC4.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
30 HP_USDHC4	Indicates the Privilege/User Mode for USDHC4.  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
29 L_USDHC3	Lock bit set by TZ software for HP_USDHC3.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
28 HP_USDHC3	Indicates the Privilege/User Mode for USDHC3  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
27 L_USDHC2	Lock bit set by TZ software for HP_USDHC2.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
26 HP_USDHC2	Indicates the Privilege/User Mode for USDHC2  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
25 L_USDHC1	Lock bit set by TZ software for HP_USDHC1.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
24 HP_USDHC1	Indicates the Privilege/User Mode for USDHC1  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
23 L_DAP	Lock bit set by TZ software for HP_DAP.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
22 HP_DAP	Indicates the Privilege/User Mode for DAP  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
21 L_ENET	Lock bit set by TZ software for HP_ENET.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
20 HP_ENET	Indicates the Privilege/User Mode for ENET  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master

*Table continues on the next page...*

## CSU\_HP0 field descriptions (continued)

Field	Description
19 L_ABPHDMA	Lock bit set by TZ software for HP_APBHDMA.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
18 HP_APBHDMA	Indicates the Privilege/User Mode for APBHDMA  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
17 L_RAWNAND	Lock bit set by TZ software for HP_RAWNAND.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
16 HP_RAWNAND	Indicates the Privilege/User Mode for RawNAND  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
15 L_CAAM	Lock bit set by TZ software for HP_CAAM.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
14 HP_CAAM	Indicates the Privilege/User Mode for CAAM  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
13 L_MLB	Lock bit set by TZ software for HP_MLB.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
12 HP_MLB	Indicate the mode Privilege/User Mode for MLB.  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
11 L_TEST	Lock bit set by TZ software for HP_TEST.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
10 HP_TEST	Indicates the Privilege/User Mode for IOMUX Test Port.  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
9 L_USB	Lock bit set by TZ software for HP_USB.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
8 HP_USB	Indicates the Privilege/User Mode for USB.  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master

*Table continues on the next page...*

**CSU\_HP0 field descriptions (continued)**

Field	Description
7 L_PU	Lock bit set by TZ software for HP_PU.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
6 HP_PU	Indicates the Privilege/User Mode for GPU3D, GPU2D, VPU, IPU1, IPU2, OpenVGEPDC, PXP, LCDIF and VDOA.  <b>NOTE:</b> IPU2 and OpenVG on i.MX6DQ only; EPDC, PXP and LCDIF on i.MX6SDL only  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
5 L_SDMA	Lock bit set by TZ software for HP_SDMA.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
4 HP_SDMA	Indicates the Privilege/User Mode for SDMA.  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
3 L_SATA	Lock bit set by TZ software for HP_SATA.  <b>NOTE:</b> This bit field is Reserved for i.MX6SDL  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
2 HP_SATA	Indicates the Privilege/User Mode for SATA  <b>NOTE:</b> This bit field is Reserved for i.MX6SDL  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master
1 L_PCIE	Lock bit set by TZ software for HP_PCIE.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
0 HP_PCIE	Indicates the Privilege/User Mode for PCIE  0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master

### 3.3.3 HP1 register (CSU\_HP1)

The SCU\_HP1 register is an expansion of the SCU\_HP0 register. See SCU\_HP0 register definition.

Address: 0h base + 204h offset = 204h

Bit	31	30	29	28	27	26	25	24		23	22	21	20	19	18	17	16
R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0
R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

#### CSU\_HP1 field descriptions

Field	Description
31–2 -	Reserved.
1 L_HDMI_HSI	Lock bit set by TZ software for HP_HDMI_HSI. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
0 HP_HDMI_HSI	Indicates the Privilege/User Mode for HDMI Tx and HSI. 0 User Mode for the corresponding master 1 Supervisor Mode for the corresponding master

### 3.3.4 Secure access register (CSU\_SA)

The Secure Access register may be programmed to specify the access policy (either Secure or Non-secure) for up to sixteen different masters. This register is used to set the access policy for Type 1 masters which are incapable of setting the policy by themselves.

The sixteen even bit positions (CSU\_SA[30,28,...,0]) in the register hold the policy indicator bits; while the odd bit positions (CSU\_SA[31,29,...,1]) contain lock bits which enable/disable writing to the corresponding policy indicator bits.

Address: 0h base + 218h offset = 218h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	-		L_HDMI_HSI	NSA_HDMI_HSI	L_USDHC4	NSA_USDHC4	L_USDHC3	NSA_USDHC3	L_USDHC2	NSA_USDHC2	L_USDHC1	NSA_USDHC1	L_DAP	NSA_DAP	L_ENET	NSA_ENET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	L_RAWNAND_APBHDMA	NSA_RAWNAND_APBHDMA	-		L_PCIE_TEST	NSA_PCIE_TEST	L_USB_MLB	NSA_USB_MLB	L_PU	NSA_PU	L_SDMA	NSA_SDMA	L_SATA	NSA_SATA	L_CP15	NSA_CP15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### CSU\_SA field descriptions

Field	Description
31–30 -	Reserved
29 L_HDMI_HSI	Lock bit set by TZ software for NSA_HDMI_HSI. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
28 NSA_HDMI_HSI	Non-Secure Access Policy indicator bit Indicate the Type (Secured/Non-Secured) Access for HDMI Tx and HSI. 0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
27 L_USDHC4	Lock bit set by TZ software for NSA_USDHC14. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
26 NSA_USDHC4	Non-Secure Access Policy indicator bit Indicate the Type (Secured/Non-Secured) Access for USDHC4. 0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master

Table continues on the next page...

## CSU\_SA field descriptions (continued)

Field	Description
25 L_USDHC3	Lock bit set by TZ software for NSA_USDHC3.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
24 NSA_USDHC3	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for USDHC3.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
23 L_USDHC2	Lock bit set by TZ software for NSA_USDHC2.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
22 NSA_USDHC2	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for USDHC2.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
21 L_USDHC1	Lock bit set by TZ software for NSA_USDHC1.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
20 NSA_USDHC1	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for USDHC1.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
19 L_DAP	Lock bit set by TZ software for NSA_DAP.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
18 NSA_DAP	Non-Secure Access Policy indicator bit ARM core platform DAP  Access Policy indicator bits  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
17 L_ENET	Lock bit set by TZ software for NSA_ENET.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
16 NSA_ENET	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for ENET.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master

Table continues on the next page...

**CSU\_SA field descriptions (continued)**

<b>Field</b>	<b>Description</b>
15 L_RAWNAND_APBHDMA	Lock bit set by TZ software for NSA_RAWNAND_APBHDMA.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
14 NSA_RAWNAND_APBHDMA	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for RawNAND and apbhdmat.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
13–12 -	Reserved
11 L_PCIE_TEST	Lock bit set by TZ software for NSA_PCIE_TEST.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
10 NSA_PCIE_TEST	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for PCIe and IOMUX Test Port.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
9 L_USB_MLB	Lock bit set by TZ software for NSA_USB_MLB.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
8 NSA_USB_MLB	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for USB and MLB.  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
7 L_PU	Lock bit set by TZ software for NSA_PU.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
6 NSA_PU	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for GPU3D, VDOA, GPU2D, IPU1, IPU2, OPENVGEPDC, PXP, LCDIF and VPU.  NOTE: IPU2 and OpenVG on i.MX6DQ only; EPDC, PXP and LCDIF on i.MX6SDL only  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
5 L_SDMA	Lock bit set by TZ software for NSA_SDMA.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
4 NSA_SDMA	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access for SDMA.

*Table continues on the next page...*

## CSU\_SA field descriptions (continued)

Field	Description
	0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
3 L_SATA	Lock bit set by TZ software for NSA_SATA.  <b>NOTE:</b> This bit field is Reserved for i.MX6SDL  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
2 NSA_SATA	Non-Secure Access Policy indicator bit  Indicates the type of access (Secured/Non-Secured) for SATA  <b>NOTE:</b> This bit field is Reserved for i.MX6SDL  0 Secure access for the corresponding Type 1 master 1 Non-secure access for the corresponding Type 1 master
1 L_CP15	Lock bit set by TZ software for NSA_CP15.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
0 NSA_CP15	Non-Secure Access Policy indicator bit  Indicate the Type (Secured/Non-Secured) Access to ARM CP15 register.  0 ARM CP15 register is accesible 1 ARM CP15 register is not accesible

## 3.3.5 HPCONTROL0 register (CSU\_HPCONTROL0)

The HP Control registers CSU\_HPCONTROL0 and CSU\_HPCONTROL1 enable CSU to control the USER/SUPERVISOR mode state for the specified masters. The register toggles the output signal csu\_hprot1 for system masters. The two possibilities sources for the csu\_hprot1 output are:

1. the hprot1 input signal, or
2. the corresponding bit in the HP register.

The even bits in the registers are used for locking the control bit values.

Address: 0h base + 358h offset = 358h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	L_USDHC4	HPC_USDHC4	L_USDHC3	HPC_USDHC3	L_USDHC2	HPC_USDHC2	L_USDHC1	HPC_USDHC1	L_DAP	HPC_DAP	L_ENET	HPC_ENET	L_ABPHDMA	HPC_APBHDMA	L_RAWNAND	HPC_RAWNAND
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	L_CAAAM	HPC_CAAAM	L_MLB	HPC_MLB	L_TEST	HPC_TEST	L_USB	HPC_USB	L_PU	HPC_PU	L_SDMA	HPC_SDMA	L_SATA	HPC_SATA	L_PCIE	HPC_PCIE
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**CSU\_HPCONTROL0 field descriptions**

Field	Description
31 L_USDHC4	Lock bit set by TZ software for HPC_USDHC4. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
30 HPC_USDHC4	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of USDHC4. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
29 L_USDHC3	Lock bit set by TZ software for HPC_USDHC3. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
28 HPC_USDHC3	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of USDHC3. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
27 L_USDHC2	Lock bit set by TZ software for HPC_USDHC2. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
26 HPC_USDHC2	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of USDHC2. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
25 L_USDHC1	Lock bit set by TZ software for HPC_USDHC1. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
24 HPC_USDHC1	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of USDHC1. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
23 L_DAP	Lock bit set by TZ software for HPC_DAP. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
22 HPC_DAP	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of DAP. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master

*Table continues on the next page...*

**CSU\_HPCONTROL0 field descriptions (continued)**

Field	Description
21 L_ENET	Lock bit set by TZ software for HPC_ENET.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
20 HPC_ENET	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of ENET.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
19 L_APBHDMA	Lock bit set by TZ software for HPC_APBHDMA.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
18 HPC_APBHDMA	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of apbhdma.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
17 L_RAWNAND	Lock bit set by TZ software for HPC_RAWNAND.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
16 HPC_RAWNAND	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of RawNAND.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
15 L_CAAM	Lock bit set by TZ software for HPC_CAAM.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
14 HPC_CAAM	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of CAAM.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
13 L_MLB	Lock bit set by TZ software for HPC_MLB.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
12 HPC_MLB	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of MLB.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master
11 L_TEST	Lock bit set by TZ software for HPC_TEST.  0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
10 HPC_TEST	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of IOMUX Test Port.  0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master

*Table continues on the next page...*

**CSU\_HPCONTROL0 field descriptions (continued)**

Field	Description
9 L_USB	<p>Lock bit set by TZ software for HPC_USB.</p> <p>0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software</p>
8 HPC_USB	<p>Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of USB.</p> <p>0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master</p>
7 L_PU	<p>Lock bit set by TZ software for HPC_PU.</p> <p>0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software</p>
6 HPC_PU	<p>Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of GPU3D, GPU2D, VPU, IPU1, IPU2, OpenVGEPDC, PXP, LCDIF and VDOA.</p> <p>NOTE: IPU2 and OpenVG on i.MX6DQ only; EPDC, PXP and LCDIF on i.MX6SDL only</p> <p>0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master</p>
5 L_SDMA	<p>Lock bit set by TZ software for HPC_SDMA.</p> <p>0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software</p>
4 HPC_SDMA	<p>Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of SDMA.</p> <p>0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master</p>
3 L_SATA	<p>Lock bit set by TZ software for HPC_SATA.</p> <p><b>NOTE:</b> This bit field is Reserved for i.MX6SDL</p> <p>0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software</p>
2 HPC_SATA	<p>Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of SATA.</p> <p><b>NOTE:</b> This bit field is Reserved for i.MX6SDL</p> <p>0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master</p>
1 L_PCIE	<p>Lock bit set by TZ software for HPC_PCIE.</p> <p>0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software</p>
0 HPC_PCIE	<p>Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of PCIE.</p> <p>0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master</p>

### 3.3.6 HPCONTROL1 register (CSU\_HPCONTROL1)

The SCU\_HPCONTROL1 register is expansion of SCU\_HPCONTROL0 register. See SCU\_HPCONTROL0 register definition.

Address: 0h base + 35Ch offset = 35Ch

Bit	31	30	29	28	27	26	25	24		23	22	21	20	19	18	17	16
R									-								
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0
R									-								
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

**CSU\_HPCONTROL1 field descriptions**

Field	Description
31–2 -	Reserved.
1 L_HDMI_HSI	Lock bit set by TZ software for HPC_HDMI_HSI. 0 No lock -- adjacent (next lower) bit can be written by software 1 Lock -- adjacent (next lower) bit cannot be written by software
0 HPC_HDMI_HSI	Determines if the Register value of the HP field corresponding will be pass as the hprot[1] of HDMI Tx and HSI. 0 Input signal hprot1 value is routed to csu_hprot1 output for the corresponding master 1 HP register bit is routed to csu_hprot1 output for the corresponding master

# Chapter 4

## System Boot

### 4.1 Overview

The boot process begins at Power On Reset (POR) where the hardware reset logic forces the ARM core to begin execution starting from the on-chip boot ROM.

Boot ROM code uses the state of the internal register BOOT\_MODE[1:0] as well as the state of various eFUSES and/or GPIO settings to determine the boot flow behavior of the device.

The main features of the ROM include:

- Support for booting from various boot devices
- Serial downloader support (USB OTG)
- Device configuration data (DCD)
- Digital signature and encryption based High Assurance Boot (HAB)
- Wake-up from low power modes
- Wake-up secondary core
- Plugin image

The boot ROM supports the following boot devices:

- NOR Flash
- NAND Flash
- OneNAND Flash
- SD/MMC
- Serial ATA (SATA) HDD (only i.MX 6Dual/6Quad)
- Serial (I2C/SPI) NOR Flash and EEPROM

In normal operation, the Boot ROM uses the state of the BOOT\_MODE register and eFUSES to determine the boot device. For development purposes, eFUSES used to determine the boot device may be overridden by using GPIO pin inputs.

## Boot modes

Boot ROM code also allows the downloading of programs to be run on the device. An example is a provisioning program that can make further use of the serial connection to provision a boot device with a new image. Typically the provisioning program is downloaded to internal RAM and allows the programming of boot devices, such as an SD/MMC Flash. The ROM Serial Downloader uses high speed USB in a non-stream mode connection.

Boot ROM allows waking up from low-power modes and waking up the secondary core of Cortex-A9. On reset the ROM checks the ARM core ID and power gating status register. On waking from low power mode, the primary core (ID=0) will skip loading an image from the boot device and jump to the address saved in PERSISTENT\_ENTRY0. The secondary cores (ID!=0) jump to the address in PERSISTENT\_ENTRY<X> where 0<X<4 in all power modes.

The device configuration data (DCD) feature allows boot ROM code to obtain SOC configuration data from an external Program Image residing on the boot device. As an example, DCD can be used to program the DDR controller for optimal settings improving the boot performance. DCD is restricted to memory areas and peripheral addresses that are considered essential for boot purposes (see [Table 4-35](#)).

A key feature of the boot ROM is the ability to perform a secure boot or High Assurance Boot (HAB). This is supported by the HAB security library which is a subcomponent of the ROM code. HAB uses a combination of hardware and software together with a Public Key Infrastructure (PKI) protocol to protect the system from executing unauthorized programs. Before the HAB allows a user's image to execute, the image must be signed. The signing process is done during the image build process by the private key holder and the signatures are then included as part of the final Program Image. If configured to do so, the ROM verifies the signatures using the public keys included in the Program Image. In addition to supporting digital signature verification to authenticate Program Images, Encrypted boot is also supported. Encrypted boot can be used to prevent cloning of the Program Image directly off the boot device. A secure boot with HAB can be performed on all boot devices supported on the chip in addition to the Serial Downloader. The HAB library in the boot ROM also provides API functions, allowing additional boot chain components (bootloaders) to extend the secure boot chain. The out-of-fab setting for SEC\_CONFIG is the Open configuration in which the ROM/HAB performs image authentication, but all authentication errors are ignored and the image is still allowed to execute.

## 4.2 Boot modes

On reset, the chip checks ARM core ID and Power Gating Controller status register.

On normal boot, core0 behavior is defined by the Boot Mode pins settings as described in [Boot mode pin settings](#). On waking up from low power boot mode, core0 skips clock settings. Boot ROM checks that PERSISTENT\_ENTRY0 (see [Persistent Bits](#)) is a pointer to valid address space (OCRAM, DDR or EIM). If PERSISTENT\_ENTRY0 is a pointer to valid range, it starts execution using entry point from PERSISTENT\_ENTRY0 register. If PERSISTENT\_ENTRY0 is a pointer to invalid range, core0 performs system reset.

For secondary cores (with ID!=0) boot ROM checks that PERSISTENT\_ENTRY is a pointer to valid address space (OCRAM, DDR or EIM). If PERSISTENT\_ENTRY is a pointer to valid range, it starts execution using entry point from PERSISTENT\_ENTRY register. If PERSISTENT\_ENTRY is a pointer to invalid range, it sets error status registers (see [Persistent Bits](#)), sends wakeup error interrupt and performs Wait For Interrupt instruction. The interrupt service routine of the other core must reconfigure the system and reset the secondary core that failed to boot.

#### **NOTE**

Code that enables the secondary cores is not part of ROM, but it must be part of upper level software.

### **4.2.1 Boot mode pin settings**

The device has four boot modes (one is reserved for Freescale use). Boot mode is selected based on the binary value stored in the internal BOOT\_MODE register.

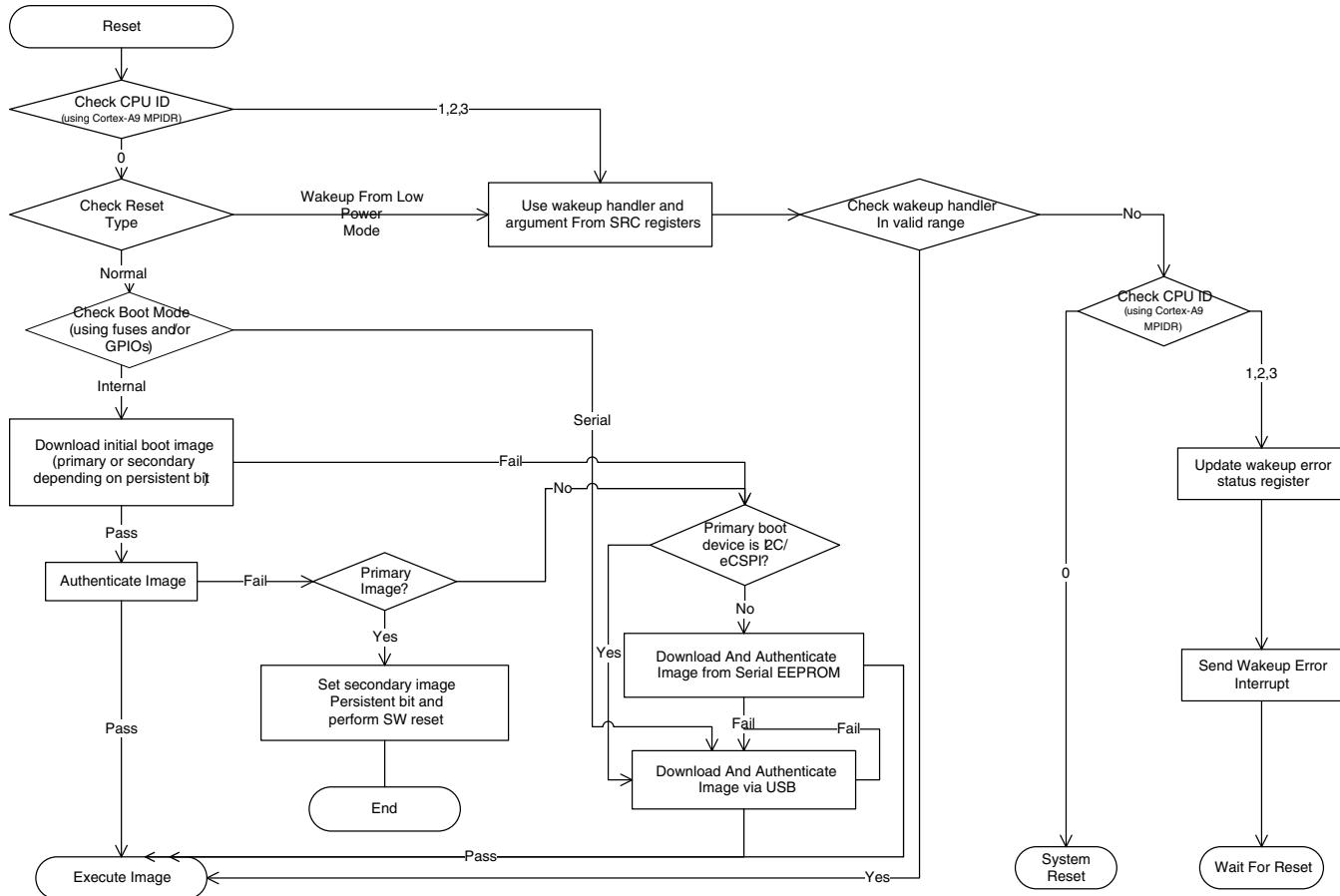
BOOT\_MODE is initialized by sampling the BOOT\_MODE0 and BOOT\_MODE1 inputs on the rising edge of POR\_B. After these inputs are sampled, their subsequent state does not affect the contents of the BOOT\_MODE internal register. The state of the internal BOOT\_MODE register may be read from the BMOD[1:0] field of the SRC Boot Mode Register (SRC\_SBMR2). The available boot modes are: Boot From Fuses, serial boot via USB, and Internal Boot. See the table below for settings.

**Table 4-1. Boot MODE Pin Settings**

BOOT_MODE[1:0]	Boot Type
00	Boot From Fuses
01	Serial Downloader
10	Internal Boot
11	Reserved

## 4.2.2 High level boot sequence

The figure below shows the high-level boot ROM code flow.



**Figure 4-1. Boot Flow**

### NOTE

For External Interface Module (EIM) boot devices, downloading initial load region to IRAM is skipped. IVT is read from EIM address space (see [Image Vector Table and Boot Data](#)). Copying initial load region and the rest of the program image is done only if the absolute start address of the image is not equal to EIM CS0 start address.

## 4.2.3 Boot From Fuses Mode (BOOT\_MODE[1:0] = 00b)

A value of 00b in the BOOT\_MODE[1:0] register selects the Boot From Fuses mode.

This mode is similar to the Internal Boot mode described in [Internal Boot Mode \(BOOT\\_MODE\[1:0\] = 0b10\)](#) with one difference. In this mode the GPIO boot override pins are ignored. The boot ROM code uses the boot eFUSE settings only. This mode also supports a secure boot using HAB.

If set to Boot From Fuses, the boot flow is controlled by the BT\_FUSE\_SEL eFUSE value. If BT\_FUSE\_SEL = 0, indicating that the boot device (for example, Flash, SD/MMC) has not yet been programmed, the boot flow jumps directly to the Serial Downloader. If BT\_FUSE\_SEL = 1, the normal boot flow is followed, where the ROM attempts to boot from the selected boot device.

The first time a board is used, the default eFUSES may be configured incorrectly for the hardware on the platform. In such a case, the Boot ROM code may try to boot from a device that does not exist. This may cause an electrical/logic violation on some pads. Using Boot From Fuses mode addresses this problem.

The first time the BT\_FUSE\_SEL = 0 eFUSE is encountered, it is not blown (thus setting BT\_FUSE\_SEL). This forces the ROM code to jump directly to the Serial Downloader. This allows a bootloader to be downloaded which can then provision the boot device with a Program Image and blow the BT\_FUSE\_SEL and the other boot configuration eFUSES. After reset, the Boot ROM code determines that BT\_FUSE\_SEL is blown (BT\_FUSE\_SEL = 1) and the ROM code performs internal boot according to the new eFUSE settings. This allows a user to set BOOT\_MODE[1:0]=00b on a production device and burn fuses on the same device (by forcing entry to the Serial Downloader), without changing the value of BOOT\_MODE[1:0] or pullups/pulldowns on the BOOT\_MODE pins.

#### 4.2.4 Serial Downloader

The Serial Downloader provides a means to download a Program Image to the chip over USB serial connection.

In this mode the ROM programs WDOG-1 for a 32-second time-out if WDOG\_ENABLE eFuse is 1, and then continuously polls for USB connection. If no activity is found on USB OTG1 and the watchdog timer expires, the ARM core is reset.

#### NOTE

The downloaded image must continue to service the watchdog timer to avoid an undesired reset from occurring.

The USB boot flow is shown in the figure below.

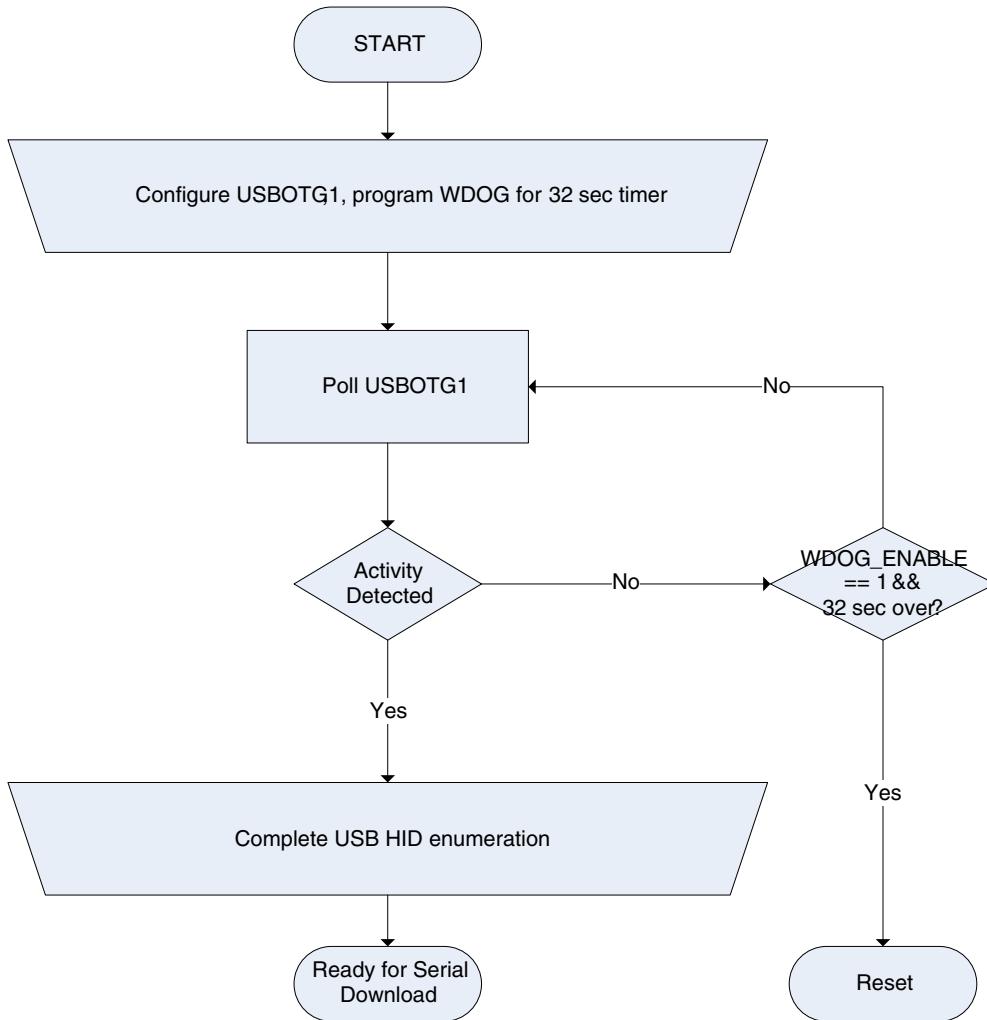


Figure 4-2. USB Boot Flow

#### 4.2.5 Internal Boot Mode (BOOT\_MODE[1:0] = 0b10)

A value of 0b10 in the BOOT\_MODE[1:0] register selects the internal boot mode. In this mode, the processor continues to execute boot code from the internal boot ROM.

The boot code performs hardware initialization, loads the Program Image from the chosen boot device, performs image validation using the HAB library (see [Boot security settings](#)), and then jumps to an address derived from the Program Image. If any error occurs during internal boot, the boot code jumps to the Serial Downloader (see [Serial Downloader](#)). A secure boot using the HAB is possible in all the three boot modes.

When set to internal boot, the boot flow may be controlled by a combination of eFUSE settings with an option of overriding the fuse settings using General Purpose I/O (GPIO) pins. The GPIO Boot Select FUSE (BT\_FUSE\_SEL) determines whether the ROM uses GPIO pins for a select number of configuration parameters or eFUSES in this mode. See [Table 4-4](#) for more details.

- If BT\_FUSE\_SEL = 1, all boot options are controlled by the eFUSES described in [Table 4-2](#).
- If BT\_FUSE\_SEL = 0, specific boot configuration parameters may be set using GPIO pins rather than eFUSES. The fuses that can be overridden when in this mode are indicated in the GPIO column of [Table 4-2](#). [Table 4-3](#) provides the details on the GPIO pins.

The use of GPIO overrides is intended for development since these pads are used for other purposes in deployed products. Freescale recommends controlling the boot configuration by eFUSES in deployed products and reserving the use of the GPIO mode for development and testing purposes only.

## 4.2.6 Boot security settings

Internal boot modes use one of three security configurations:

- Closed: This level is intended for use with shipping secure products. All HAB functions are executed and security hardware is initialized (the Security Controller, or SNVS, enters Secure state), DCD is processed if present, and the program image is authenticated by HAB prior to its execution. All detected errors will be logged, and the boot flow aborted with control passing to the serial downloader. At this level, execution does not leave the internal ROM unless the target executable image has been authenticated.
- Open: This level is intended for use in non-secure products or during the development phases of a secure product. All HAB functions are executed as for a closed device. Security hardware is initialized (except the SNVS is left in Non-Secure state), DCD is processed if present, and the program image is authenticated by HAB prior to its execution. All detected errors will be logged, but have no influence on the boot flow, which continues as if the errors did not occur. This configuration is useful for secure product development, since the Program Image will run even if the authentication data is missing or incorrect, and the error log can be examined to determine the cause of authentication failure.
- Field Return: This level is intended for parts returned from shipped products.

**NOTE**

If the DIR\_BT\_DIS eFuse is not blown, authentication may be bypassed. In this case the system is not secure.

## 4.3 Device Configuration

This section describes the external inputs that control the behavior of the Boot ROM code.

This includes boot device selection (SPI, EIM, NOR, SD, MMC, etc.), boot device configuration (SD bus width, speed, etc), and so on. In general, the source for this configuration comes from eFUSES embedded inside the chip. However, certain configuration parameters can be sourced from GPIO pins allowing further flexibility during the development process.

### 4.3.1 Boot eFUSE Descriptions

The table below is a comprehensive list of the configuration parameters that the ROM uses.

**Table 4-2. Boot eFUSE Descriptions**

Fuse	Config uratio n	Definition	GPIO	Shipped Value	Settings
DIR_BT_DIS	OEM	Disables Freescale reserved modes. Must be set for secure boot.	NA	0	0 Reserved Freescale modes enabled 1 Reserved Freescale modes disabled
BT_FUSE_SEL	OEM	In internal Boot mode BOOT_MODE[1:0] = 10, the BT_FUSE_SEL fuse determines whether the boot settings indicated by a Yes in the GPIO column are controlled by GPIO pins or eFUSE settings in the On-Chip OTP Controller (OCOTP).  In Boot From Fuse mode BOOT_MODE[1:0] = 00, BT_FUSE_SEL fuse indicates whether bit configuration eFuses have been programmed.	NA	0	If BOOT_MODE[1:0] = 0b10 0 Bits of SBMR are overridden by GPIO pins.  1 Specific bits of SBMR are controlled by eFUSE settings.  If BOOT_MODE[1:0] = 0b00 0 BOOT configuration eFuses are not yet programmed. Boot flow jumps to serial downloader.  1 BOOT configuration eFuses have been programmed. Regular boot flow is performed.

*Table continues on the next page...*

**Table 4-2. Boot eFUSE Descriptions  
(continued)**

Fuse	Config uratio n	Definition	GPIO	Shipped Value	Settings
SEC_CONFIG[1:0]	SEC_C ONFIG[ 0] - Freesc ale  SEC_C ONFIG[ 1] - OEM	Security Configuration as defined in <a href="#">Boot security settings</a>	NA	01	00 Reserved  01 Open (allows any program image, even if authentication fails)  1x Closed (Program image executes only if authenticated)
FIELD_RETURN	OEM	Enables Freescale reserved modes			0 - Freescale reserved modes are enabled/disabled based on DIR_BT_DIS value  1 - Freescale reserved modes are enabled
SRK_HASH[255:0]	OEM	256-bit hash value of super root key (SRK_HASH)	NA	0	Settings vary - used by HAB
DIE-X-CORDINATE[7:0]  DIE-Y-CORDINATE[7:0]  WAFER_NO[4:0]  LOT_NO_ENC[42:40]  LOT_NO_ENC[39:32]  LOT_NO_ENC[31:24]  LOT_NO_ENC[23:16]  LOT_NO_ENC[15:8]  LOT_NO_ENC[7:0]	Freesc ale	Device Unique ID, 64-bit UID.	NA	Unique ID	Settings vary - used by HAB
BT_MMU_DISABLE (BOOT_CFG3[6])	OEM	MMU/L1 D Cache/PL310 disable bit used by boot ROM for fast HAB processing	Yes	0	0 - MMU/L1 D Cache/PL310 is enabled by ROM during the boot  1 - MMU/L1 D Cache/PL310 is disabled by ROM during the boot
L1 I-Cache DISABLE (BOOT_CFG3[7])	OEM	L1 I Cache disable bit used by boot during entire execution	Yes	0	0 - L1 I Cache is enabled by ROM during the boot  1 - L1 I Cache is disabled by ROM during the boot
BT_FREQ (BOOT_CFG3[2])	OEM	Frequency Selection	Yes	0	0 - ARM - 792MHz, DDR - 532MHz, AXI - 264MHz  1 - ARM - 396MHz, DDR - 352MHz, AXI - 176MHz
BOOT_CFG1[7:0]	OEM	Boot Configuration1	Yes	0	Specific to selected boot mode
BOOT_CFG2[7:0]	OEM	Boot Configuration2	Yes	0	Specific to selected boot mode
BOOT_CFG4[6:0]	OEM	Boot Configuration4		0	Specific to selected boot mode

Table continues on the next page...

**Table 4-2. Boot eFUSE Descriptions  
(continued)**

Fuse	Config uratio n	Definition	GPIO	Shipped Value	Settings
BOOT_CFG4[7]	OEM	Infinite Loop Enable at start of boot ROM. Used for debugging purposes. Ignored if DIR_BT_DIS is 1 and FIELD_RETURN is 0.	Yes	0	0 - Disabled 1 - Enabled
LPB_BOOT	OEM	USB Low Power Boot	No	0	00 - LPB Disable 01 - 1 GPIO (default frequencies) 10 - Divide by 2 11 - Divide by 4
BT_LPB_POLARITY	OEM	USB Low Power Boot GPIO polarity	No	0	0 - 0 on GPIO pad indicates low power condition 1 - 1 on GPIO pad indicates low power condition
WDOG_ENABLE	OEM	Watchdog reset counter enable	No	0	0 - watchdog reset counter is disabled during serial downloader 1 - watchdog reset counter is enabled during serial downloader
MMC_DLL_DLY[6:0]	OEM	USDHC Delay Line settings	No	0	USDHC Delay Line settings
SRK_REVOCODE[2:0]	OEM	SRK revocation mask	No	0	SRK revocation mask
DISABLE_SDMMC_MFG	OEM	Disable SDMMC manufacture mode	Yes	0	0: enable SD/MMC MFG mode 1: disable SD/MMC MFG mode
PAD_SETTINGS	OEM	Override values for SD/MMC and NAND boot modes	No	0	Override the following IO PAD settings: PAD_SETTINGS[0] - Slew Rate PAD_SETTINGS[3:1] Drive Strength PAD_SETTINGS[5:4] - Speed Settings .

### 4.3.2 GPIO Boot Overrides

The table below provides a list of GPIO boot overrides.

**Table 4-3. GPIO Override Contact Assignments**

Package Pin	Direction on reset	eFuse
BOOT_MODE1	Input	Boot Mode Selection
BOOT_MODE0	Input	
EIM_DA0	Input	BOOT_CFG1[0]
EIM_DA1	Input	BOOT_CFG1[1]

*Table continues on the next page...*

**Table 4-3. GPIO Override Contact Assignments (continued)**

Package Pin	Direction on reset	eFuse
EIM_DA2	Input	BOOT_CFG1[2]
EIM_DA3	Input	BOOT_CFG1[3]
EIM_DA4	Input	BOOT_CFG1[4]
EIM_DA5	Input	BOOT_CFG1[5]
EIM_DA6	Input	BOOT_CFG1[6]
EIM_DA7	Input	BOOT_CFG1[7]
EIM_DA8	Input	BOOT_CFG2[0]
EIM_DA9	Input	BOOT_CFG2[1]
EIM_DA10	Input	BOOT_CFG2[2]
EIM_DA11	Input	BOOT_CFG2[3]
EIM_DA12	Input	BOOT_CFG2[4]
EIM_DA13	Input	BOOT_CFG2[5]
EIM_DA14	Input	BOOT_CFG2[6]
EIM_DA15	Input	BOOT_CFG2[7]
EIM_A16	Input	BOOT_CFG3[0]
EIM_A17	Input	BOOT_CFG3[1]
EIM_A18	Input	BOOT_CFG3[2]
EIM_A19	Input	BOOT_CFG3[3]
EIM_A20	Input	BOOT_CFG3[4]
EIM_A21	Input	BOOT_CFG3[5]
EIM_A22	Input	BOOT_CFG3[6]
EIM_A23	Input	BOOT_CFG3[7]
EIM_A24	Input	BOOT_CFG4[0]
EIM_WAIT	Input	BOOT_CFG4[1]
EIM_LBA	Input	BOOT_CFG4[2]
EIM_EB0	Input	BOOT_CFG4[3]
EIM_EB1	Input	BOOT_CFG4[4]
EIM_RW	Input	BOOT_CFG4[5]
EIM_EB2	Input	BOOT_CFG4[6]
EIM_EB3	Input	BOOT_CFG4[7]

The input pins provided are sampled at boot, and can be used to override corresponding eFUSE values, depending on the setting of the BT\_FUSE\_SEL fuse. Table below describes boot options control sampling in different boot modes.

**Table 4-4. Boot Options Control Selection**

BOOT_MODE[1:0]	BT_FUSE_SEL Value	Boot Options Controlled By
00	0	eFUSES
	1	

*Table continues on the next page...*

**Table 4-4. Boot Options Control Selection  
(continued)**

BOOT_MODE[1:0]	BT_FUSE_SEL Value	Boot Options Controlled By
01	0	GPIO pins
	1	eFUSES
10	0	GPIO pins
	1	eFUSES

### 4.3.3 Device Configuration Data

DCD is configuration information contained in a Program Image, external to the ROM, that the ROM interprets to configure various on-chip peripherals. See [Device Configuration Data \(DCD\)](#) for more details on Device Configuration Data.

## 4.4 Device Initialization

This section describes the details on the ROM and provides initialization details.

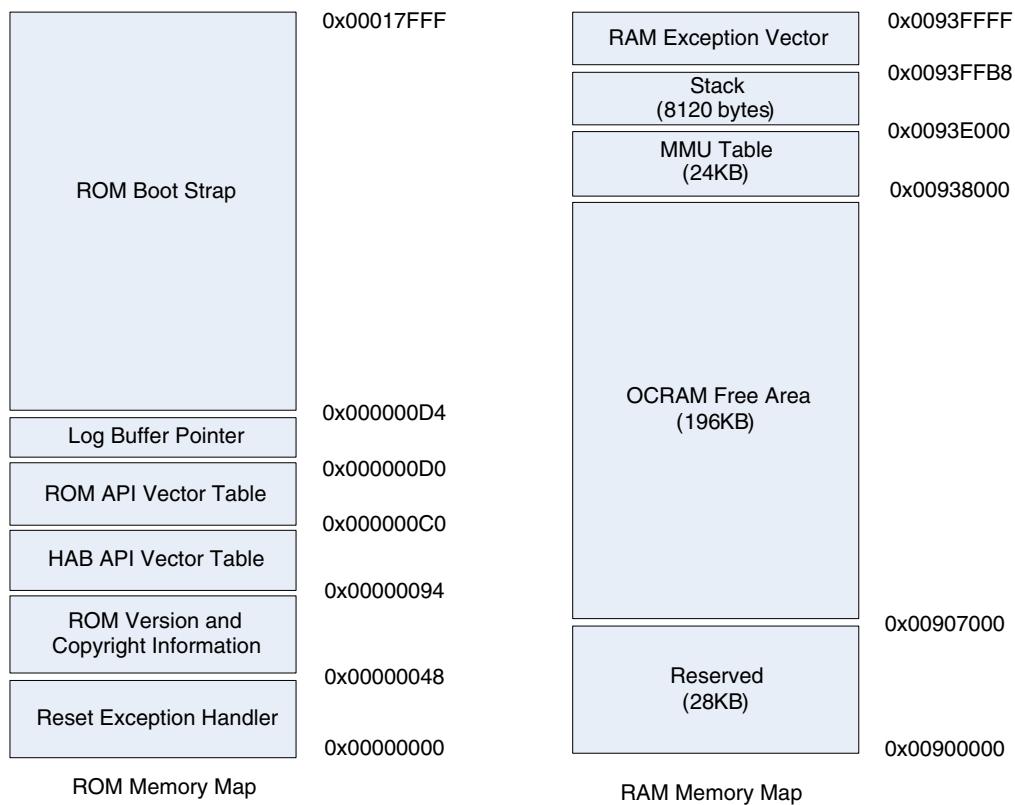
This includes details on:

- The ROM Memory Map
- The RAM Memory Map
- On-chip blocks that the ROM should make use of or change POR register default values
- Clock initialization
- Enabling the MMU/L2 cache when performing a secure boot (SEC\_CONFIG = Closed)
- Exception handling and interrupt handling

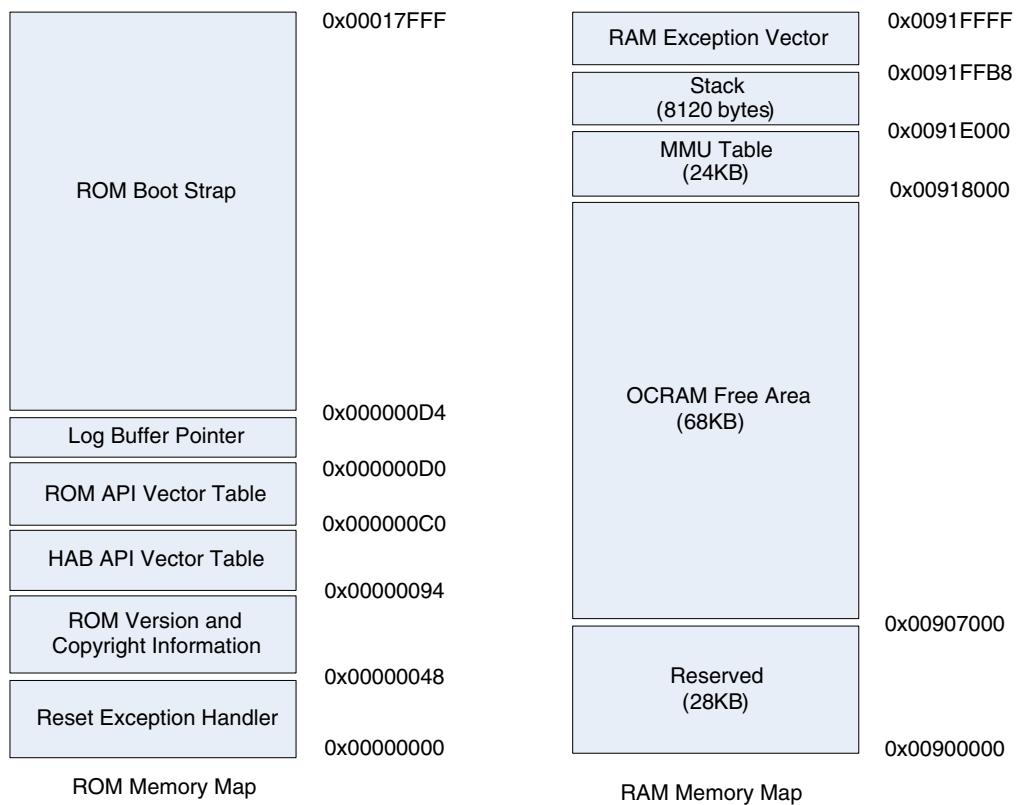
### 4.4.1 Internal ROM /RAM memory map

[Figure 4-3](#) shows the iROM memory map for the i.MX 6Dual/6Quad.

[Figure 4-4](#) shows the iROM memory map for the i.MX 6Solo/6DualLite.



**Figure 4-3. Internal ROM and RAM memory map for i.MX 6Dual/6Quad**

**Figure 4-4. Internal Rom and Ram memory map for i.MX 6Solo/6DualLite****NOTE**

The entire OCRAM region can be used freely post boot.

#### 4.4.2 Boot Block Activation

The boot ROM affects a number of different hardware blocks which are activated and play a vital role in the boot flow.

The ROM configures and uses the following blocks (listed in alphabetical order) during the boot process. Note that the blocks actually used depend on the boot mode and boot device selection:

- APBH - DMA engine to drive the GPMI module
- BCH - 40-bit error correction hardware engine with AXI bus master and private connection to GPMI
- CCM - Clock Control Module
- ECSPI - Enhanced Configurable Serial Peripheral Interface
- EIM - External Interface Module. Used for NOR and OneNAND devices

- I2C - I2C Controller
- GPMI - NAND controller pin interface
- OCOTP\_CTRL - On-Chip OTP Controller. The OCOTP contains the eFUSES.
- IOMUXC - I/O Multiplexer Control allows GPIO use to override eFUSE boot settings
- CAAM - Cryptographic Acceleration and Assurance Module
- SNVS - Secure Non-Volatile Storage
- SRC - System Reset Controller
- USB - Used for serial download of a boot device provisioning program
- USDHC - Ultra Secure Digital Host Controller
- WDOG-1 - Watchdog Timer
- DTCP - Digital Transmission Content Protection
- HDCP - High-bandwidth Digital Content Protection

#### 4.4.3 Clocks at Boot Time

The table below shows the various clocks and their sources used by ROM.

**Table 4-5. Normal Frequency Clocks Configuration**

Clock	CCM signal	Source	Frequency(MHz) BT_FREQ=0	Frequency(MHz) BT_FREQ=1
System PLL	pll1_sw_clk		792	792
528MHz PLL	pll2_sw_clk		528	528
480MHz PLL	pll3_sw_clk		480	480
ARM core clock	arm_clk_root	System PLL	792	396
EIM	emi_slow_clk_root	528MHz PLL	132	132
AHB	ahb_clk_root	528MHz PLL/PFD352	132	76.6
IPG	ipg_clk_root	528MHz PLL/PFD352	66	38.3
AXI_A	axi_a	528MHz PLL/PFD352	528396	306.6
AXI_B	axi_b	528MHz PLL/PFD352	264	153.3
USB	usboh3_clk_root	480MHz PLL	60	60
USDHC	usdhc1_clk_root usdhc2_clk_root usdhc3_clk_root usdhc4_clk_root	PFD400	198	198
ECSPI	ecspi_clk_root	480MHz PLL	60	60
I2C	per_clk_root	528MHz PLL	66	38.3

## Device Initialization

Following reset, each ARM core has access to all peripherals. The ROM code will disable the clocks listed in the following table, except for the boot devices listed in the second column.

**Table 4-6. List Of Disabled Clocks**

Clock Name	Enabled For Boot Device
CCGR0_APBHDMA	
CCGR1_ECSPI1	ECSPI1
CCGR1_ECSPI2	ECSPI2
CCGR1_ECSPI3	ECSPI3
CCGR1_ECSPI4	ECSPI4
CCGR1_FEC	
CCGR1_EPIT1	
CCGR1_EPIT2	
CCGR2_I2C1_SERIAL	I2C1
CCGR2_I2C2_SERIAL	I2C2
CCGR2_I2C3_SERIAL	I2C3
CCGR3_OPENVGAXICLK	
CCGR4_PWM1	
CCGR4_PWM2	
CCGR4_PWM3	
CCGR4_PWM4	
CCGR5_SDMA	
CCGR5_SPDIF	
CCGR5_SSI1	
CCGR5_SSI2	
CCGR5_SSI3	
CCGR5_UART	
CCGR5_UART_SERIAL	
CCGR6_USBOH3	USB
CCGR6_USDHC1	USDHC1
CCGR6_USDHC2	USDHC2
CCGR6_USDHC3	USDHC3
CCGR6_USDHC4	USDHC4
CCGR6_EMI_SLOW	NOR, OneNAND

#### 4.4.4 Enabling MMU and Caches

The boot ROM includes a feature of enabling the Memory Management Unit (MMU) and caches to improve boot speed when performing a secure boot with SEC\_CONFIG=Closed ([High Assurance Boot \(HAB\)](#)). L1 instruction cache is enabled at the start of image download. L1 data cache, L2 cache and MMU are enabled during image authentication. Once HAB authentication completes the ROM disables the L2 cache and MMU.

L1 Instruction cache is controlled by L1 I-Cache eFuse. By default the L1 I-Cache is enabled.

The MMU feature is controlled by the BT\_MMU\_DISABLE eFUSE. By default the BT\_MMU\_DISABLE eFUSE is not blown meaning the ROM uses MMU, L1 Data and L2 caches of the ARM core. ROM establishes a page table in the MMU Page Table area of iRAM. The ROM also configures the L1 and L2 cache as write through. This improves the performance of the HAB signature verification software.

Enabling the MMU when booting non-securely with SEC\_CONFIG=Open, and setting the CSF pointer in the Image Vector Table to NULL, has no impact on the boot performance. With this configuration it is recommended to blow BT\_MMU\_DISABLE fuse.

#### 4.4.5 Exception Handling

The exception vectors located at the start of ROM are used to map all the ARM exceptions (except the reset exception) to a duplicate exception vector table in internal RAM.

During the boot phase of CPU0, the RAM vectors point to the serial downloader in ROM.

During the boot phase of a secondary CPU, the internal RAM vectors point to a function that sets the error status registers (see [Persistent Bits](#)), sends a wakeup error interrupt and performs the Wait For Interrupt instruction. The interrupt service routine of primary CPU must reconfigure the system and reset the secondary CPU.

After boot the program image can overwrite the vectors as required. The code shown below is used to map the ROM exception vector table to the duplicate one in RAM.

##### Mapping ROM Exception Vector Table

```
; Define linker area for ROM exception vector table
AREA IROM_VECTORS, CODE, READONLY
LDR    PC, Reset_Addr
LDR    PC, Undefined_Addr
LDR    PC, SWI_Addr
```

## Device Initialization

```
LDR      PC, Prefetch_Addr
LDR      PC, Abort_Addr
NOP          ; Reserved vector
LDR      PC, IRQ_Addr
LDR      PC, FIQ_Addr
LDR      PC, SW_monitor_addr
;; Define exception vector table
Reset_Addr    DCD      start_address
Undefined_Addr DCD      iRAM_Undefined_Handler
SWI_Addr       DCD      iRAM_SWI_Handler
Prefetch_Addr  DCD      iRAM_Prefetch_Handler
Abort_Addr     DCD      iRAM_Abort_Handler
                DCD      0           ; Reserved vector
IRQ_Addr       DCD      iRAM_IRQ_Handler
FIQ_Addr       DCD      iRAM_FIQ_Handler
SW_monitor_add DCD      SW monitor exception
start_address  DCD start ;reset handler vector
```

## 4.4.6 Interrupt Handling During Boot

No special interrupt handling routines are required during the boot process. Interrupts are disabled during boot ROM execution and may be enabled in a later boot stage.

## 4.4.7 Persistent Bits

Some modes of boot ROM require registers that keep their values after warm reset. SRC General Purpose registers are used for this purpose.

See the table below for persistent bits list and description.

**Table 4-7. Persistent Bits**

Bit Name	Bit Location	Description
PERSIST_SECONDARY_BOOT	SRC_GPR10[30]	This bit identifies which image must be used - primary and secondary. Used only for boot modes that support redundant boot.
PERSIST_BLOCK_REWRITE	SRC_GPR10[29]	This bit is used as warning. It identifies that there are errors in NAND blocks that hold the application image. See <a href="#">NAND Flash</a> for more details.
PERSISTENT_ENTRY0[31:0]	SRC_GPR1[31:0]	Holds entry function for CPU0 for waking-up from low power mode.
PERSISTENT_ARG0[31:0]	SRC_GPR2[31:0]	Holds argument of entry function for CPU0 for waking-up from low power mode.
PERSISTENT_ENTRY1[31:0]	SRC_GPR3[31:0]	Holds entry function for CPU1.
PERSISTENT_ARG1[31:0]	SRC_GPR4[31:0]	Holds argument of entry function for CPU1.
PERSISTENT_ENTRY2[31:0]	SRC_GPR5[31:0]	Holds entry function for CPU2 (i.MX 6Quad only).
PERSISTENT_ARG2[31:0]	SRC_GPR6[31:0]	Holds argument of entry function for CPU2 (i.MX 6Quad only).
PERSISTENT_ENTRY3[31:0]	SRC_GPR7[31:0]	Holds entry function for CPU3 (i.MX 6Quad only).

*Table continues on the next page...*

**Table 4-7. Persistent Bits (continued)**

Bit Name	Bit Location	Description
PERSISTENT_ARG3[31:0]	SRC_GPR8[31:0]	Holds argument of entry function for CPU3 (i.MX 6Quad only).
CPU3_ERROR_STATUS	SRC_GPR10[27]	CPU3 error status bit (i.MX 6Quad only)
CPU2_ERROR_STATUS	SRC_GPR10[26]	CPU2 error status bit (i.MX 6Quad only)
CPU1_ERROR_STATUS	SRC_GPR10[25]	CPU1 error status bit

## 4.5 Boot Devices (Internal Boot)

The Chip supports the following boot Flash devices:

- NOR Flash with External Interface Module (EIM), located on CS0, 16-bits and 32-bit bus width
- OneNAND Flash with EIM interface, located on CS0, 16-bits bus width
- Raw NAND (MLC and SLC), and Toggle-mode NAND flash through GPMI-2 interface. Page sizes of 2 Kbyte, 4 Kbyte and 8 Kbyte. Bus widths of 8-bit with 2 through 40-bit BCH Hardware ECC (Error Correction) supported.
- SD/MMC/eSD/SDXC/eMMC4.4 via USDHC interface, supporting high capacity cards
- EEPROM boot via SPI (serial flash) and I2C (via ECSPI and I2C blocks respectively)
- Serial ATA (SATA) boot via SATA interface

The selection of external boot device type is controlled by BOOT\_CFG1[7:4] eFUSES. See the table below for more details.

**Table 4-8. Boot Device Selection**

BOOT_CFG1[7:4]	Boot Device
0000	NOR/OneNAND (EIM)
0001	Reserved
0010	SSD/Hard Disk (SATA)
0011	Serial ROM (I2C/SPI)
010x	SD/eSD/SDXC
011x	MMC/eMMC
1xxx	NAND

## 4.5.1 NOR Flash/OneNAND using EIM Interface

The External Interface Module (EIM) works in the asynchronous mode, and supports either muxed, Address/Data, or non-muxed schemes based on fuse settings.

**Table 4-9. EIM Boot eFUSE Descriptions**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[7:4]	OEM	Boot Device Selection	Yes	0000	0000 - Boot from EIM Interface
BOOT_CFG1[3]	OEM	NOR/OneNAND Selection	Yes	0	0 - NOR 1 - OneNAND
BOOT_CFG2[7:6]	OEM	Muxing Scheme	Yes	00	00 - Muxed, 16-bit data (low half) interface 01 - Not muxed, 16-bit data (high half) interface 10 - Not muxed, 16-bit data (low half) interface 11 - Reserved
BOOT_CFG2[5:4]	OEM	OneNAND Page Size	Yes	00	00 - 1K 01 - 2K 10 - 4K 11 - Reserved

### 4.5.1.1 NOR Flash Boot Operation

Booting from the NOR Flash is supported via EIM interface. The ROM reads Image Vector Table and Boot Data structures to determine if the image can be executed directly from EIM address space or should be copied to other memory.

The start field of Boot Data Structure specifies the final location of the image (see [Image Vector Table and Boot Data](#)).

### 4.5.1.2 OneNAND Flash Boot Operation

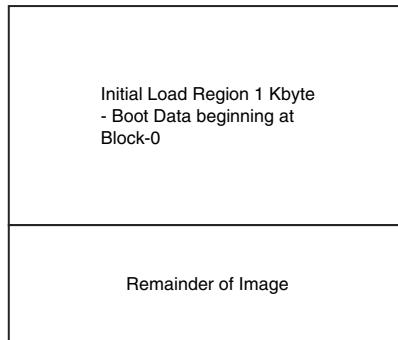
At system power-up, the OneNAND device automatically copies an Initial Load Region of 1 Kbyte from the start of the flash array (sector 0 and sector 1, page 0, block 0) to its Boot RAM (OneNAND's internal RAM).

**NOTE**

The OneNAND boot RAM memory containing the Initial 1K Load Region must contain the IVT, DCD and the Boot Data structures.

Next, the ROM processes the DCD and then proceeds to copy the Program Image contents to the application destination pointer (located in the start entry of Boot Data (see [Image Vector Table and Boot Data](#)). The ROM determines the size of the Program Image by the length specified by size entry in Boot Data structure (see [Image Vector Table and Boot Data](#)). A failure loading data from the OneNAND device for any reason forces the Chip to enter the Serial Downloader, otherwise the booting from the OneNAND device continues.

The figure below illustrates the layout of the Program Image on a OneNAND boot device.



**Figure 4-5. Program Image Layout on a OneNAND Flash Device**

Prior to accessing the OneNAND device, the Chip waits approximately 500  $\mu$ s after Power On Reset. This delay is required for the OneNAND device to become ready. After this initial 500  $\mu$ s delay it can take an addition 70  $\mu$ s for the OneNAND device to load the Initial Load Region of 1 Kbytes into its boot RAM. The Chip polls the OneNAND device Interrupt Status Register to confirm that the first 1 Kbytes has been loaded to the OneNAND boot RAM before continuing with the boot flow.

#### 4.5.1.3 IOMUX Configuration for EIM Devices

The EIM interface uses dedicated contacts on the IC.

## Boot Devices (Internal Boot)

The contacts assigned to the data signals used by EIM are shown in the table below.

**Table 4-10. EIM IOMUX Pin Configuration**

Signal	A/D16 (Muxed, 16-bit data low half interface)	A+DH (Not muxed, 16-bit data high half interface)	A+DL (Not muxed, 16-bit data low half interface)
<b>DATA0</b>	EIM_DA0.alt0	EIM_D16.alt0	CSI0_DATA_EN.alt1
<b>DATA1</b>	EIM_DA1.alt0	EIM_D17.alt0	CSI0_VSYNC.alt1
<b>DATA2</b>	EIM_DA2.alt0	EIM_D18.alt0	CSI0_DAT4.alt1
<b>DATA3</b>	EIM_DA3.alt0	EIM_D19.alt0	CSI0_DAT5.alt1
<b>DATA4</b>	EIM_DA4.alt0	EIM_D20.alt0	CSI0_DAT6.alt1
<b>DATA5</b>	EIM_DA5.alt0	EIM_D21.alt0	CSI0_DAT7.alt1
<b>DATA6</b>	EIM_DA6.alt0	EIM_D22.alt0	CSI0_DAT8.alt1
<b>DATA7</b>	EIM_DA7.alt0	EIM_D23.alt0	CSI0_DAT9.alt1
<b>DATA8</b>	EIM_DA8.alt0	EIM_D24.alt0	CSI0_DAT12.alt1
<b>DATA9</b>	EIM_DA9.alt0	EIM_D25.alt0	CSI0_DAT13.alt1
<b>DATA10</b>	EIM_DA10.alt0	EIM_D26.alt0	CSI0_DAT14.alt1
<b>DATA11</b>	EIM_DA11.alt0	EIM_D27.alt0	CSI0_DAT15.alt1
<b>DATA12</b>	EIM_DA12.alt0	EIM_D28.alt0	CSI0_DAT16.alt1
<b>DATA13</b>	EIM_DA13.alt0	EIM_D29.alt0	CSI0_DAT17.alt1
<b>DATA14</b>	EIM_DA14.alt0	EIM_D30.alt0	CSI0_DAT18.alt1
<b>DATA15</b>	EIM_DA15.alt0	EIM_D31.alt0	CSI0_DAT19.alt1
<b>ADDR0</b>		EIM_DA0.alt0	
<b>ADDR1</b>		EIM_DA1.alt0	
<b>ADDR2</b>		EIM_DA2.alt0	
<b>ADDR3</b>		EIM_DA3.alt0	
<b>ADDR4</b>		EIM_DA4.alt0	
<b>ADDR5</b>		EIM_DA5.alt0	
<b>ADDR6</b>		EIM_DA6.alt0	
<b>ADDR7</b>		EIM_DA7.alt0	
<b>ADDR8</b>		EIM_DA8.alt0	
<b>ADDR9</b>		EIM_DA9.alt0	
<b>ADDR10</b>		EIM_DA10.alt0	
<b>ADDR11</b>		EIM_DA11.alt0	
<b>ADDR12</b>		EIM_DA12.alt0	
<b>ADDR13</b>		EIM_DA13.alt0	
<b>ADDR14</b>		EIM_DA14.alt0	
<b>ADDR15</b>		EIM_DA15.alt0	

## 4.5.2 NAND Flash

The boot ROM supports a number of MLC/SLC NAND Flash devices from different vendors and LBA NAND Flash devices. The Error Correction and Control (ECC) subblock (BCH40) is used to detect the errors.

### 4.5.2.1 NAND eFUSE Configuration

The boot ROM determines the configuration of external the NAND flash by parameters, either provided by eFUSE, or sampled on GPIO pins, during boot.

See the table below for parameters details.

**Table 4-11. NAND Boot eFUSE Descriptions**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[7]	OEM	Boot Device Selection	Yes	0	1 - Boot from NAND Interface
BOOT_CFG1[5]	OEM	BT_TOGGLEMODE	Yes	0	0 - raw NAND 1 - toggle mode NAND
BOOT_CFG1[4]	OEM	Override Pad Settings	Yes	0	0 - Use default values 1 - Use PAD_SETTINGS values
BOOT_CFG1[3:2]	OEM	Number of devices	Yes	00	00 - 1 device 01- 2 device 10- 4 device 11- Reserved
BOOT_CFG1[1:0]	OEM	Address Cycles	Yes	00	00 - 3 01 - 2 10 - 4 11 - 5
BOOT_CFG2[7:5]	OEM	Toggle Mode 33MHz Preamble Delay, Read Latency	Yes	000	000 - 16 GPMICLK cycles. 001 - 1 GPMICLK cycles 010 - 2 GPMICLK cycles 011 - 3 GPMICLK cycles 100 - 4 GPMICLK cycles 101 - 5 GPMICLK cycles 110 - 6 GPMICLK cycles 111 - 7 GPMICLK cycles
BOOT_CFG2[4:3]	OEM	Boot Search Count	Yes	00	00 - 2 01 - 2 10 - 4 11 - 8

*Table continues on the next page...*

**Table 4-11. NAND Boot eFUSE Descriptions  
(continued)**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG2[2:1]	OEM	Pages In Block	Yes	00	00 - 128 01 - 64 10 - 32 11 - 256
BOOT_CFG2[0]	OEM	Reset time	Yes	0	0 - 12ms 1 - 22ms (LBA NAND)

#### 4.5.2.2 NAND Flash Boot Flow and Boot Control Blocks (BCB)

There are two BCB data structures: FCB and DBBT.

As part of the NAND media initialization, the ROM driver uses safe NAND timings to search for a Firmware Configuration Block (FCB) that contains the optimum NAND timings, page address of Discovered Bad Block Table (DBBT) Search Area and start page address of primary and secondary firmware.

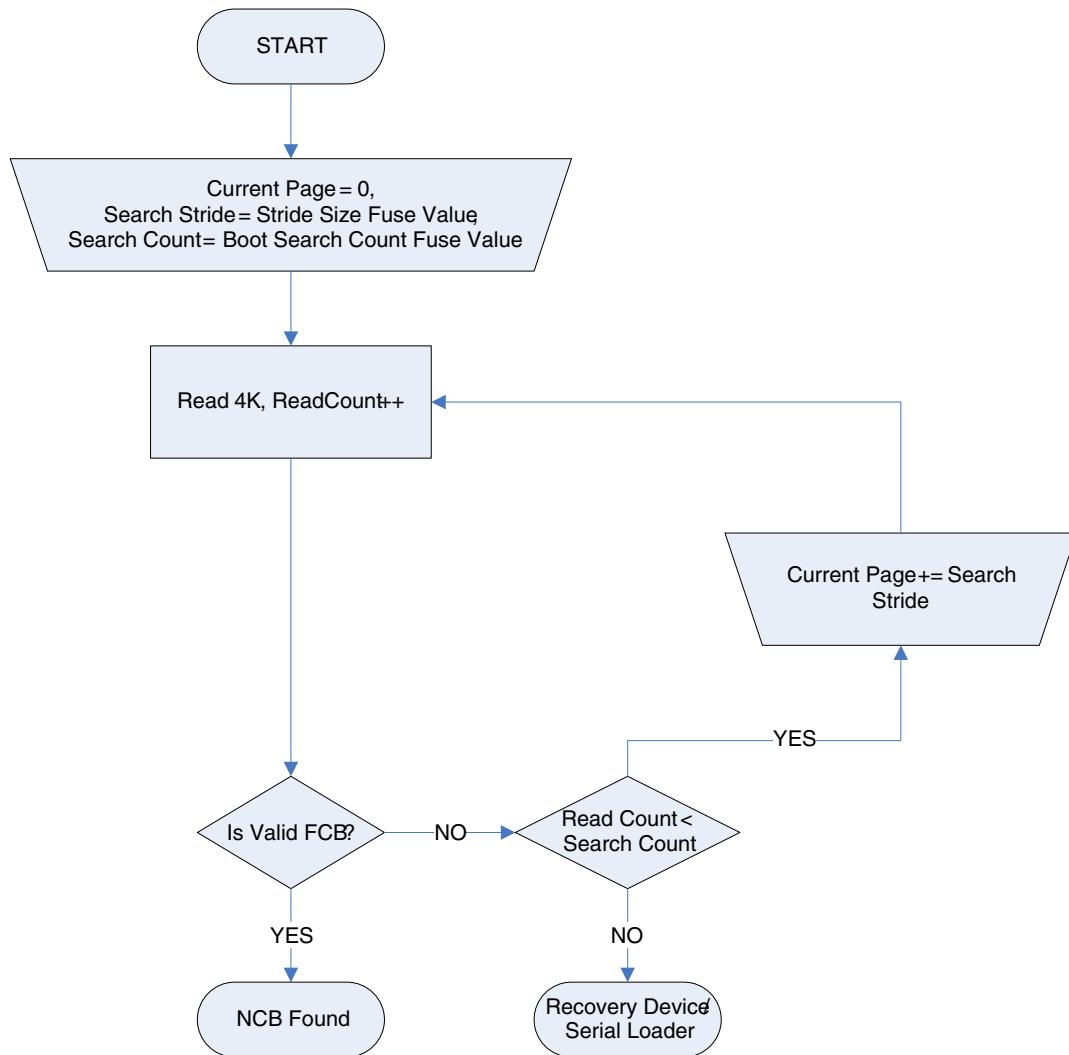
The hardware ECC level to use is embedded inside FCB block. The FCB data structure is protected using software ECC (SEC-DED Hamming Codes). Driver reads raw 2112 bytes of first sector and runs through software ECC engine that determines whether FCB data is valid or not.

If the FCB is found, the optimum NAND timings are loaded for further reads. If the ECC fails, or the fingerprints do not match, the Block Search state machine increments page number to Search Stride number of pages to read for the next BCB until SearchCount pages have been read.

If search fails to find a valid FCB, the NAND driver responds with an error and the boot ROM enters into serial download mode.

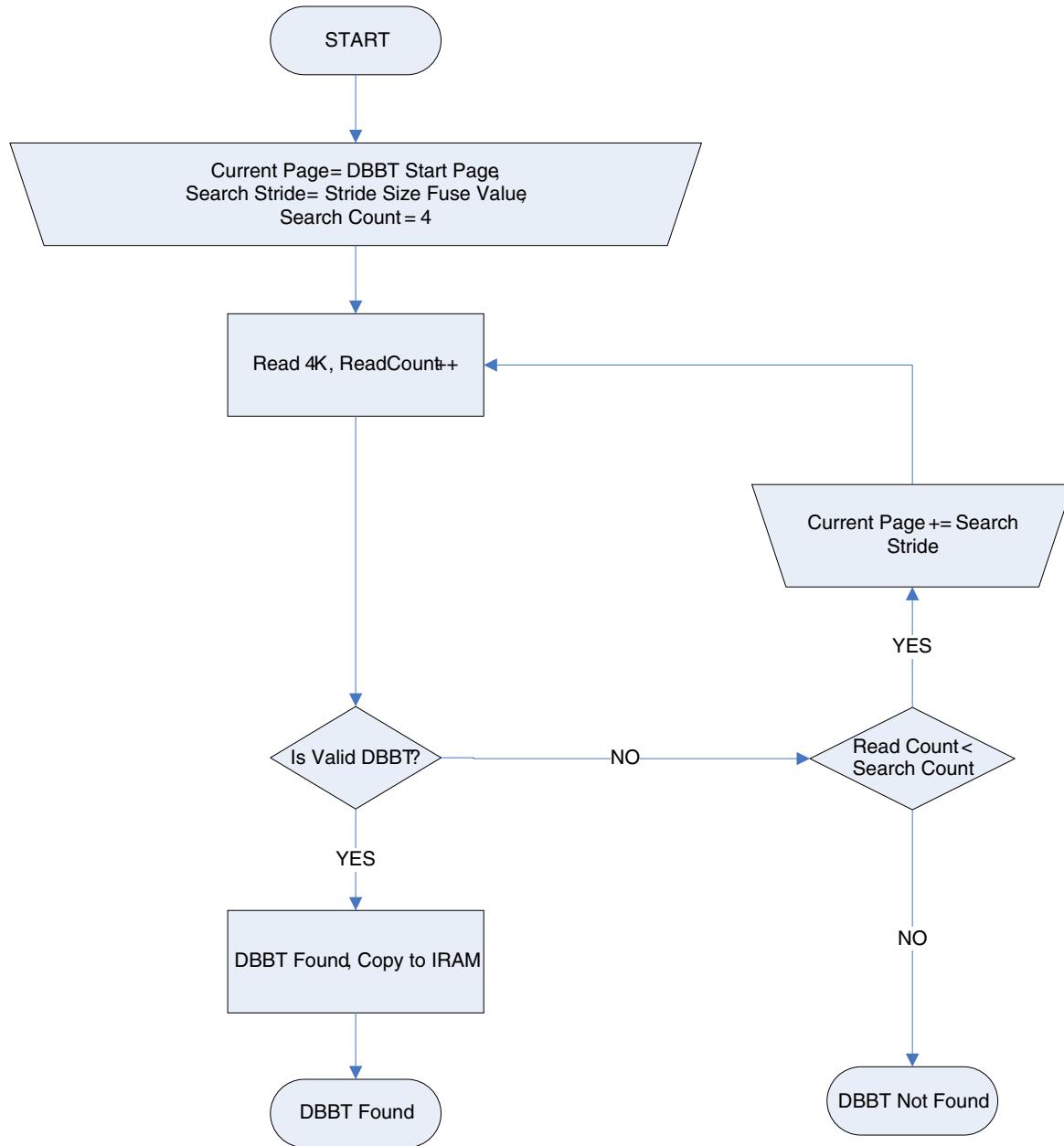
The FCB contains the page address of DDBT Search Area, and the page address for primary and secondary boot images. DDBT is searched in DDBT Search Area just like how FCB is searched. After the FCB is read, the DDBT is loaded, and the primary or secondary boot image is loaded using starting page address from FCB.

The state diagram of FCB search is shown in the following figure.

**Figure 4-6. FCB Search Flow**

Once FCB is found, the boot ROM searches for the Discovered Bad Blocks Table (DBBT). If DBBT Search Area is 0 in FCB, then ROM assumes that there are no bad blocks on NAND device. See the figure below for the DBBT search flow.

## Boot Devices (Internal Boot)



**Figure 4-7. DBBT Search Flow**

The BCB search and load function also monitors the ECC correction threshold and sets the PERSIST\_BLOCK\_REWRITE persistent bit if the threshold exceeds the maximum ECC correction ability.

If during primary image read there is a page with a number of errors higher than ECC can correct, the boot ROM will turn on PERSIST\_SECONDARY\_BOOT bit and perform SW reset. (After SW reset, secondary image will be used.)

If during secondary image read there is a page with number of errors higher than ECC can correct, the boot ROM will go to serial loader.

### 4.5.2.3 Firmware Configuration Block

The FCB is the first sector in the first good block. The FCB should be present at each search stride of the search area.

The search area contains copies of the FCB at each stride distance, so in case the first NAND block becomes corrupted, the ROM will find its copy in the next NAND block. The search area should span over at least two NAND blocks. The location information for DDBT search area, FW1, and FW2 are all specified in the FCB. Flash Control Block Structure is as shown in the table below.

**Table 4-12. Flash Control Block Structure**

Name	Start Byte	Size in Bytes	Description
Reserved	0	4	Reserved for Fingerprint #1(Checksum)
FingerPrint	4	4	32 bit word with a value of 0x4E434220, in ascii "FCB"
Version	8	4	32-bit version number; this version of FCB is 0x00000001
m_NANDTiming	12	8	8 bytes of data for 8 NAND Timing Parameters from NAND datasheet. The 8 parameters are: m_NandTiming[0]=data_setup, m_NandTiming[1]=data_hold, m_NandTiming[2]=address_setup, m_NandTiming[3]=dsample_time, m_NandTiming[4]=nand_timing_state, m_NandTiming[5]=REA, m_NandTiming[6]=RLOH, m_NandTiming[7]=RHOH. ROM only uses first 4 parameters but FCB provides space for other 4 parameters to be used by boot-loader or other applications.
PageDataSize	20	4	Number of bytes of data in a page. Typically, this is 2048 bytes for 2112 bytes page size or 4096 bytes for 4314/4224 bytes page size or 8192 for 8568 bytes page size
TotalPageSize	24	4	Total number of bytes in page. Typically, 2112 for 2 KB page or 4224 or 4314 for 4 KB page or 8568 for 8 KB page.
SectorsPerBlock	28	4	Number of pages per block. Typically 64 or 128 or depending on NAND device type.

*Table continues on the next page...*

**Table 4-12. Flash Control Block Structure  
(continued)**

Name	Start Byte	Size in Bytes	Description
NumberOfNANDs	32	4	Not used by ROM
TotalInternalDie	36	4	Not used by ROM
CellType	40	4	Not used by ROM
EccBlockNEccType	44	4	Value from 0 to 16 used to set BCH Error Correction level 0, 2, 4, .. or 40 for Block BN of ECC page, used in configuring BCH40 page layout registers
EccBlock0Size	48	4	Size of block B0, used in configuring BCH40 page layout registers
EccBlockNSize	52	4	Size of block BN, used in configuring BCH40 page layout registers
EccBlock0EccType	56	4	Value from 0 to 16 used to set BCH Error Correction level 0, 2, 4, .. or 40 for Block BN of ECC page, used in configuring BCH40 page layout registers
MetadataBytes	60	4	Size of metadata bytes used in configuring BCH40 page layout registers
NumEccBlocksPerPage	64	4	Number of ECC blocks BN not including B0. This value is used in configuring BCH40 page layout registers
EccBlockNEccLevelSDK	68	4	Not used by ROM
EccBlock0SizeSDK	72	4	Not used by ROM
EccBlockNSizeSDK	76	4	Not used by ROM
EccBlock0EccLevelSDK	80	4	Not used by ROM
NumEccBlocksPerPageSDK	84	4	Not used by ROM
MetadataBytesSDK	88	4	Not used by ROM
EraseThreshold	92	4	Not used by ROM
Firmware1_startingPage	104	4	Page number address where first copy of bootable firmware is located
Firmware2_startingPage	108	4	Page number address where second copy of bootable firmware is located
PagesInFirmware1	112	4	Size of first copy of firmware in pages
PagesInFirmware2	116	4	Size of second copy of firmware in pages
DBBTSearchAreaStartAddress	120	4	Page address for bad block table search area
BadBlockMarkerByte	124	4	This is an input offset in BCH page for ROM to swap with first byte of metadata after reading a page using BCH40. ROM supports restoration of manufacturer marked bad block markers in the page and this offset is the bad block marker offset location
BadBlockMarkerStartBit	128	4	This is an input bit offset in BadBlockMarkerByte for ROM to use when swapping 8 bits with first byte of metadata.
BBMarkerPhysicalOffset	132	4	This is the offset where manufacturer leaves bad block marker on a page

*Table continues on the next page...*

**Table 4-12. Flash Control Block Structure  
(continued)**

Name	Start Byte	Size in Bytes	Description
BCHType	136	4	0 for BCH20 and 1 for BCH40. The Chip is backward compatible to BCH20 and this field tell ROM to use BCH20 or BCH40 block
TMTiming2_ReadLatency	140	4	Toggle mode NAND timing parameter read latency, ROM use this value to configure timing2 register of GPMI
TMTiming2_PreambleDelay	144	4	Toggle mode NAND timing parameter Preamble Delay. ROM use this value to configure timing2 register of GPMI
TMTiming2_CEDelay	148	4	Toggle mode NAND timing parameter CE Delay. ROM use this value to configure timing2 register of GPMI
TMTiming2_PostambleDelay	152	4	Toggle mode NAND timing parameter Postamble Delay. ROM use this value to configure timing2 register of GPMI
TMTiming2_CmdAddPause	156	4	Toggle mode NAND timing parameter Cmd Add Pause. ROM use this value to configure timing2 register of GPMI
TMTiming2_DataPause	160	4	Toggle mode NAND timing parameter Data Pause. ROM use this value to configure timing2 register of GPMI
TMSpeed	164	4	This is the toggle mode speed for ROM to configure gpmi clock. 0 for 33 MHz, 1 for 40 MHz and 2 for 66 MHz
TMTiming1_BusyTimeout	168	4	Toggle mode NAND timing parameter Busy Timeout. ROM use this value to configure timing1 register of GPMI
DISBBM	172	4	If 0 ROM will swap BadBlockMarkerByte with metadata[0] after reading a page using BCH40. If the value set is 1 then ROM will not do swapping
BBMark_spare_offset	176	4	The offset in mata data place which stores the data in Bad block marker place.
Onfi_sync_enable	180	4	Enable the Onfi nand sync mode support
Onfi_sync_speed	184	4	Speed for onfi nand sync mode: 0 - 24MHZ, 1 - 33MHZ, 2 - 40MHZ, 3 - 50MHZ, 4 - 66MHZ, 5 - 80MHZ, 6 - 100MHZ, 7 - 133MHZ, 8 - 160MHZ, 9 - 200MHZ
Onfi_syncNANDData	188	28	parameters for onfi nand sync mode timing. They are read latency, ce_delay, preamble_delay, postamble_delay, cmdadd_pause, data_pause, busy_timeout
DISBB_Search	216	4	Disable the badblock search function when reading the firmware, only using DBBT.

#### 4.5.2.4 Discovered Bad Block Table

See the table below for DBBT format.

**Table 4-13. DBBT Structure**

Name	Start Byte	Size in Bytes	Description
reserved	0	4	-
FingerPrint	4	4	32-bit word with a value of 0x44424254,in ascii "DBBT"
Version	8	4	32-bit version number; this version of DBBT is 0x00000001
reserved	12	4	-
DBBT_NUM_OF_PAGES	16	4	Size of DBBT in pages
reserved	20	4*PageSize-20	-
reserved	4*PageSize	4	-
Number of Entries	4*PageSize + 4	4	Number of bad blocks
Bad Block Number	4*PageSize + 8	4	First bad block number
Bad Block Number	4*PageSize + 12	4	Second bad block number
...-	-	-	...next bad block number
...-	-	-	...-
Last bad block number	-	-	last bad block number

#### 4.5.2.5 Bad Block Handling in the ROM

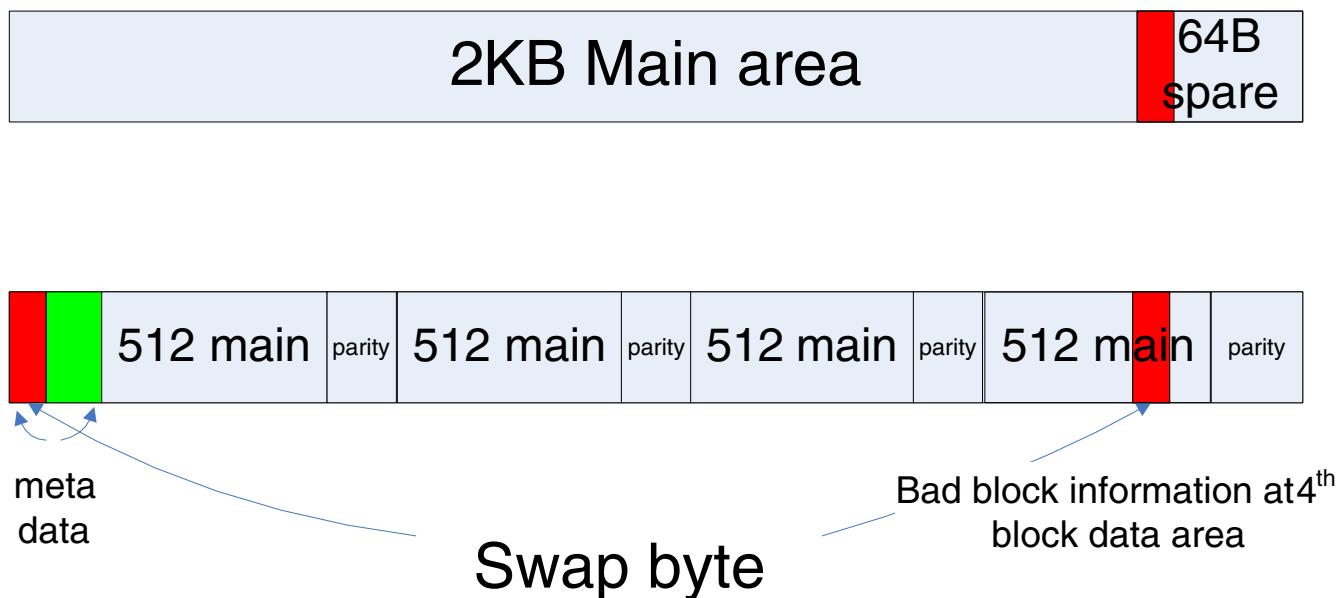
During firmware boot, at the block boundary, the Bad Block table is searched for a match to the next block.

If no match is found, the next block can be loaded. If a match is found, the block must be skipped and the next block checked.

If Bad Block table start page is null, check the manufactory made Bad Block marker. The location of Bad Block maker is at the first 3 or last 3 pages in every block of the NAND flash. NAND manufacturers normally use one byte in the spare area of certain pages within a block to mark a block is bad or not. 0xFF means good block, non FF means bad block.

In order to preserve the BI (bad block information), flash updater or gang programmer applications need to swap Bad Block Information (BI) data to byte 0 of metadata area for every page before programming NAND flash. ROM when loading firmware copies back the value at metadata[0] to BI offset in page data. The figure below shows how the factory bad block marker is preserved.

Bad block information at column address 2048



**Figure 4-8. Factory Bad Block Marker Preservation**

In the FCB structure, there are two elements `m_u32BadBlockMarkerByte` and `m_u32BadBlockMarkerStartBit` to indicate the byte and bit place in the page data, that manufacturer marked the bad block marker.

#### 4.5.2.6 Toggle Mode DDR NAND Boot

If BT\_TOGGLEMODE efuse is blown then ROM does the following to boot from Samsung's toggle mode DDR NAND.

##### 4.5.2.6.1 GPMI and BCH Clocks Configuration

ROM sets the clock source and the dividers in CCM registers. If `BOOT_CFG1[5]` is set(toggle mode), GPMI/BCH CLK source is PLL2PFD4, and running at 66MHZ, otherwise GPMI/ BCH CLK souce is PLL3, running at 24 Mhz. ROM sets default values to timing0, timing1 and timing2 gpmi registers for 24 MHz clock speed.

It uses fuse `BOOT_CFG2[7:5]` to configure GPMI timing2 register parameters preamble delay and read latency, the default value for these parameters is 2 when fuses are not blown.

Default timing parameter values used by ROM for toggle-mode device:

- Timing0.ADDRESS\_SETUP = 5

#### Boot Devices (Internal Boot)

- Timing0.DATA\_SETUP = 10
- Timing0.DATA\_HOLD = 10
- Timing1.DEVICE\_BUSY\_TIMEOUT = 0 x 500
- Timing2.READ\_LATENCY = BOOT\_CFG2[7:5] if blown, otherwise 2
- Timing2.CE\_DELAY = 2
- Timing2.PREAMBLE\_DELAY = BOOT\_CFG2[7:5] if blown, otherwise 2
- Timing2.POSTAMBLE\_DELAY = 3
- Timing2.CMDADD\_PAUSE = 4
- Timing2.DATA\_PAUSE = 6

Default timing parameters can be overriden by TMTiming2\_ReadLatency, TMTiming2\_PreambleDelay, TMTiming2\_CEDelay, TMTiming2\_PostambleDelay, TMTiming2\_CmdAddPause, TMTiming2\_DataPause parameters of FCB.

#### 4.5.2.6.2 Setup DMA for DDR Transfers

In DMA descriptors GPMI is configured to read page data at double data rate, the word length is set to 16 and transfer count to half of page size.

#### 4.5.2.6.3 Reconfigure Timing and Speed Using Values in FCB

After reading FCB page with GPMI set to default timings and speed 33 MHz, ROM reconfigures CCM dividers to run gpmi/bch clks to desired speed specified in FCB for rest of boot process. The GPMI timing registers are also reconfigured to values specified in FCB.

The GPMI speed can be configured using FCB parameter TMSpeed: 0 - 24MHZ, 1 - 33MHZ, 2 - 40MHZ, 3 - 50MHZ, 4 - 66MHZ, 5 - 80MHZ, 6 - 100MHZ, 7 - 133MHZ, 8 - 160MHZ, 9 - 200MHZ.

The GPMI timing0 register fields data\_setup, data\_hold and address\_setup are set to values specified for data\_setup, data\_hold and address\_setup in FCB member m\_NANDTiming.

The GPMI timing1.DEVICE\_BUSY\_TIMEOUT is set to value specified in FCB member TMTiming1\_BusyTimeout.

The GPMI timing2 register values are set using FCB members TMTiming2.READ\_LATENCY, CE\_DELAY, PREAMBLE\_DELAY, POSTAMBLE\_DELAY, CMDADD\_PAUSE and DATA\_PAUSE.

#### 4.5.2.7 Typical NAND Page Organization

#### 4.5.2.7.1 BCH ECC Page Organization

The first data block is called block 0 and the rest of the blocks are called block N. Separate ECC levels can be used for block 0 and block N.

The metadata bytes should be located at the beginning of a page, starting at byte 0, followed by data block 0, followed by ECC bytes for data block 0, followed by block 1 and its ECC bytes, and so on until N data blocks. The ECC level for block 0 can be different from the ECC level of rest of the blocks.

For NAND boot, with page size restrictions and data block size restricted to 512 bytes, only few combinations of ECC for block 0 and block N are possible.

The figure below shows the valid layout for 2112 byte sized page.

M	Block0 512 bytes	EccB0	Block1 512 bytes	EccBN	Block2 512 bytes	EccBN	Block3 512 bytes	EccBN
---	---------------------	-------	---------------------	-------	---------------------	-------	---------------------	-------

**Figure 4-9. Valid Layout for 2112 bytes Sized Page**

The example below is for 13 bits of parity(GF13). The number of ECC bits required for a data block is calculated using (ECC\_Correction\_Level \* 13) bits.

In the above layout the ECC size for EccB0 and EccBN should be selected to not exceed a total page size of 2112 bytes. EccB0 and EccBN can be one of 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 bits ECC correction level. The total bytes would then be:

$$[M + (\text{data\_block\_size} \times 4) + ([\text{EccB0} + (\text{EccBN} \times 3)] \times 13) / 8] \leq 2112;$$

M = metadata bytes and data\_block\_size is 512.

There are 4 data blocks of 512 bytes each in a page of 2k page sized NAND. The values of EccB0 and EccBN should be such that the above calculation would not result in a value greater than 2112 bytes.

M	Block0 512 bytes	EccB0	Block1 512 bytes	EccBN	Block2 512 bytes	EccBN	Block3 512 bytes	EccBN
	Block4 512 bytes	EccBN	Block5 512 bytes	EccBN	Block6 512 bytes	EccBN	Block7 512 bytes	EccBN

**Figure 4-10. Valid Layout for 4 Kbytes Sized Page**

Different NAND manufacturers have different sizes for a 4K page; 4314 bytes is typical.

$$[M + (\text{data\_block\_size} \times 8) + ([\text{EccB0} + (\text{EccBN} \times 7)] \times 13) / 8] \leq 4314;$$

M= metadata bytes and data\_block\_size is 512.

There are 8 data blocks of 512 bytes each in a page of a 4k page sized NAND. The values of EccB0 and EccBN should be such that above calculation should not result in a value greater than the size of a page in a 4k page NAND.

#### 4.5.2.7.2 Metadata

The number of bytes used for metadata is specified in FCB. Metadata for BCH encoded pages will be placed at the beginning of a page. ROM only cares about the first byte of metadata to swap it with bad block marker byte in page data after each page read; it is important to have at least one byte for the metadata bytes field in FCB data structure.

#### 4.5.2.8 IOMUX Configuration for NAND

The table below shows the RawNAND IOMUX pin configuration.

**Table 4-14. NAND IOMUX Pin Configuration**

Signal	Pad Name
CLE	NANDF_CLE.alt0
ALE	NANDF_ALE.alt0
WP_N	NANDF_WP_B.alt0
RD_N	SD4_CMD.alt1
WR_N	SD4_CLK.alt1
READY0	NANDF_RB0.alt0
DQS	SD4_DAT0.alt2
CE0N	NANDF_CS0.alt0
CE1N	NANDF_CS1.alt0

Table continues on the next page...

**Table 4-14. NAND IOMUX Pin Configuration (continued)**

CE2N	NANDF_CS2.alt0
CE3N	NANDF_CS3.alt0
D0	NANDF_D0.alt0
D1	NANDF_D1.alt0
D2	NANDF_D2.alt0
D3	NANDF_D3.alt0
D4	NANDF_D4.alt0
D5	NANDF_D5.alt0
D6	NANDF_D6.alt0
D7	NANDF_D7.alt0

### 4.5.3 Expansion Device

The ROM supports booting from MMC/eMMC and SD/eSD compliant devices.

#### 4.5.3.1 Expansion Device eFUSE Configuration

SD/MMC/eSD/eMMC/SDXC boot can be performed using either USDHC-1, USDHC-2, USDHC-3, or USDHC-4 ports, based on setting of the BOOT\_CFG2[4:3] (Port Select) fuse or it's associated GPIO input value at boot. All USDHC ports support eMMC4.3 and eMMC4.4 fast boot.

See the table below for details.

**Table 4-15. USDHC Boot eFUSE Descriptions**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[7:6]	OEM	Boot Device Selection	Yes	00	01 - Boot from USDHC Interfaces
BOOT_CFG1[5]	OEM	SD/MMC Selection	Yes	0	0 - SD/eSD/SDXC 1 - MMC/eMMC
BOOT_CFG1[4]	OEM	Fast Boot Support	Yes	0	0 - Normal Boot 1 - Fast Boot

*Table continues on the next page...*

**Table 4-15. USDHC Boot eFUSE Descriptions  
(continued)**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[3:2]	OEM	SD/MMC Speed Mode	Yes	00	MMC 0x - High Speed Mode 1x - Normal Speed Mode SD 0x - High/Normal 10 - SDR50 11 - SDR104
BOOT_CFG1[1]	OEM	SD Power Cycle Enable/ eMMC Reset Enable	Yes	0	MMC 0 - eMMC reset disabled 1 - eMMC reset enabled via SD_RST pad (on USDHC3 and USDHC4 only) SD 0 - No power cycle 1 - Power cycle enabled via SD_RST pad (on USDHC3 and USDHC4 only)
BOOT_CFG1[0]	OEM	SD Loopback Clock Source Sel(for SDR50 and SDR104 only)	Yes	00	0 - through SD pad 1 - direct
BOOT_CFG2[7:5]	OEM	Bus Width/SD Calibration Step	Yes	000	SD/eSD/SDXC (BOOT_CFG1[5]=0) Bus Width xx0 - 1-bit xx1 - 4-bit SD Calibration Step 00x - 1 delay cells 01x - 1 delay cells 10x - 2 delay cells 11x - 3 delay cells MMC/eMMC (BOOT_CFG1[5]=1) 000 - 1-bit 001 - 4-bit 010 - 8-bit 101 - 4-bit DDR (MMC 4.4) 110 - 8-bit DDR (MMC 4.4) Else - reserved.

*Table continues on the next page...*

**Table 4-15. USDHC Boot eFUSE Descriptions  
(continued)**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG2[4:3]	OEM	Port Select	Yes	00	00 - USDHC-1 01 - USDHC-2 10 - USDHC-3 11 - USDHC-4
BOOT_CFG2[2]	OEM	DLL Override (eMMC Only)	Yes	0	0 - Boot ROM default. 1 - Apply value per fuse field MMC_DLL_DLY[6:0]
BOOT_CFG2[1]	OEM	Boot Acknowledge Disable / Pull-Down During Power Cycle Enable	Yes	0	MMC 0 - Boot Acknowledge Enabled. 1 - Boot Acknowledge Disabled. SD 0 - Use default SD pad settings during power cycle 1 - Set pull-down on SD pads during power cycle (used only if "SD Power Cycle Enable" enabled)
BOOT_CFG2[0]	OEM	Override Pad Settings	Yes	0	0 - Use default values 1 - Use PAD_SETTINGS values
MMC_DLL_DLY[6:0]	OEM	MMC DLL Value / UHSI Calibaration Start Value	No	0000000	MMC DLL Value / UHSI Calibaration Start Value

Boot code supports following standards.

- MMCv4.4 or less
- eMMCv4.4 or less
- SDv2.0 or less
- eSDv2.10 rev-0.9, with or without FAST\_BOOT.
- SDXCv3.0

MMC/SD/eSD/SDXC/eMMC can be connected to any of USDHC-1,2,3,4 blocks and can be booted by copying 4Kbyte of data from MMC/SD/eSD/eMMC device to internal RAM. After checking the Image Vector Table header value (0xD1) from Program Image, the ROM code performs a DCD check. After successful DCD extraction, the ROM code extracts from Boot Data Structure the destination pointer and length of image to be copied to RAM device from where code execution occurs.

The maximum image size to load in SD/MMC boot is 32MB. This is due to the limited number of uSDHC ADMA Buffer Descriptors allocated by ROM.

**NOTE**

The Initial 4Kbyte of Program Image must contain the IVT, DCD and the Boot Data structures.

**Table 4-16. SD/MMC Frequencies**

	SD	MMC	MMC (DDR Mode)
Identification (KHz)	347.22		
Normal Speed Mode (MHz)	25	20	25
High Speed Mode (MHz)	50	40	50
UHSI SDR50 (MHz)	100		
UHSI SDR104 (MHz)	200		

**NOTE**

The boot ROM code reads application image length and application destination pointer from image.

**4.5.3.2 MMC and eMMC Boot**

The following table provides MMC and eMMC boot details.

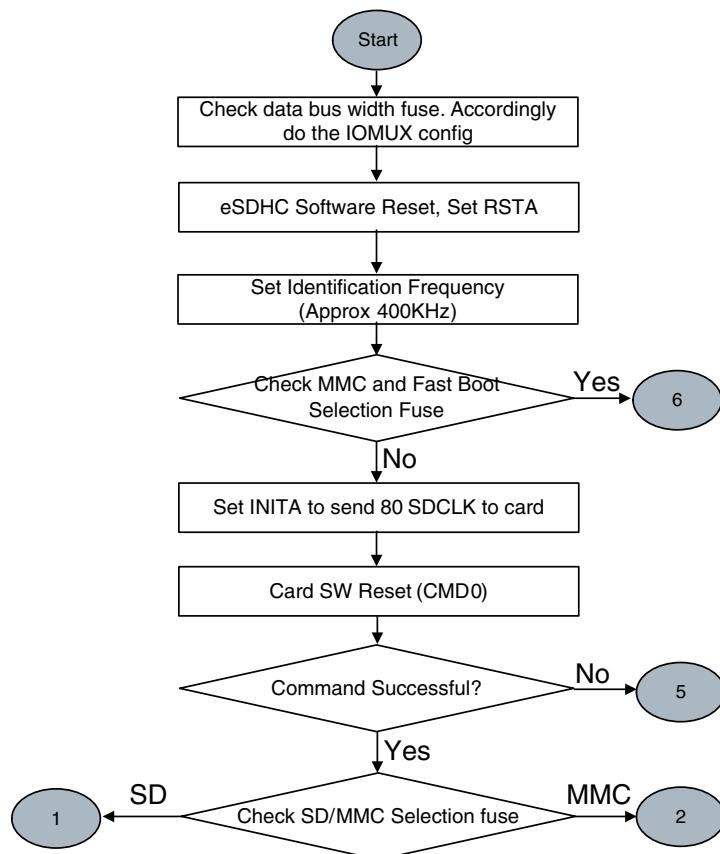
**Table 4-17. MMC and eMMC Boot Details**

Normal Boot Mode	<p>During initialization (normal boot mode) the MMC frequency is set to 347.22 KHz. When the MMC card enters the identification portion of the initialization, voltage validation is performed and the ROM boot code checks high voltage settings and card capacity. The ROM boot code supports both high capacity and low capacity MMC/eMMC cards. After initialization phase is complete, the ROM boot code switches to a higher frequency (20 MHz in Normal boot mode or 40MHz in High Speed mode). eMMC is also interfaced via USDHC and follows the same flow as MMC.</p> <p>The boot partition can be selected for an MMC4.x card after the card initialization is complete. The ROM code reads the BOOT_PARTITION_ENABLE field in the Ext_CSD[179] to get the boot partition to be set. If there is no boot partition mentioned in BOOT_PARTITION_ENABLE field or the user partition has been mentioned, ROM boots from the user partition.</p>
------------------	--

*Table continues on the next page...*

**Table 4-17. MMC and eMMC Boot Details (continued)**

eMMC4.3 or eMMC4.4 Device Supporting Special Boot Mode	If using an eMMC4.3 or eMMC4.4 device supporting special boot mode, it can be initiated by pulling the CMD line low. If BOOT ACK is enabled, the eMMC4.3/eMMC4.4 device sends the BOOT ACK via DATA lines and ROM can read the BOOT ACK [S010E] to identify the eMMC4.3/eMMC4.4 device. eMMC4.3/eMMC4.4 device with "Boot mode" feature can only be supported via ESDHCV3-3 and with or without BOOT ACK. If BOOT ACK is enabled ROM waits 50 ms to get the BOOT ACK and if BOOT ACK is received by ROM. If BOOT ACK is disabled ROM waits 1 second for data. If BOOT ACK or data was received then eMMC4.3/eMMC4.4 is booted in "Boot mode", otherwise eMMC4.3/eMMC4.4 boots as a normal MMC card from the selected boot partition. This boot mode can be selected by BOOT_CFG1[4] (Fast Boot) fuse. BOOT ACK is selected by BOOT_CFG2[1].
eMMC4.4 Device	If using eMMC4.4 device, Double Data Rate (DDR) mode can be used. This mode can be selected by BOOT_CFG2[7:5] (Bus Width) fuse.

**Figure 4-11. Expansion Device Boot Flow (1 of 6)**

## Boot Devices (Internal Boot)

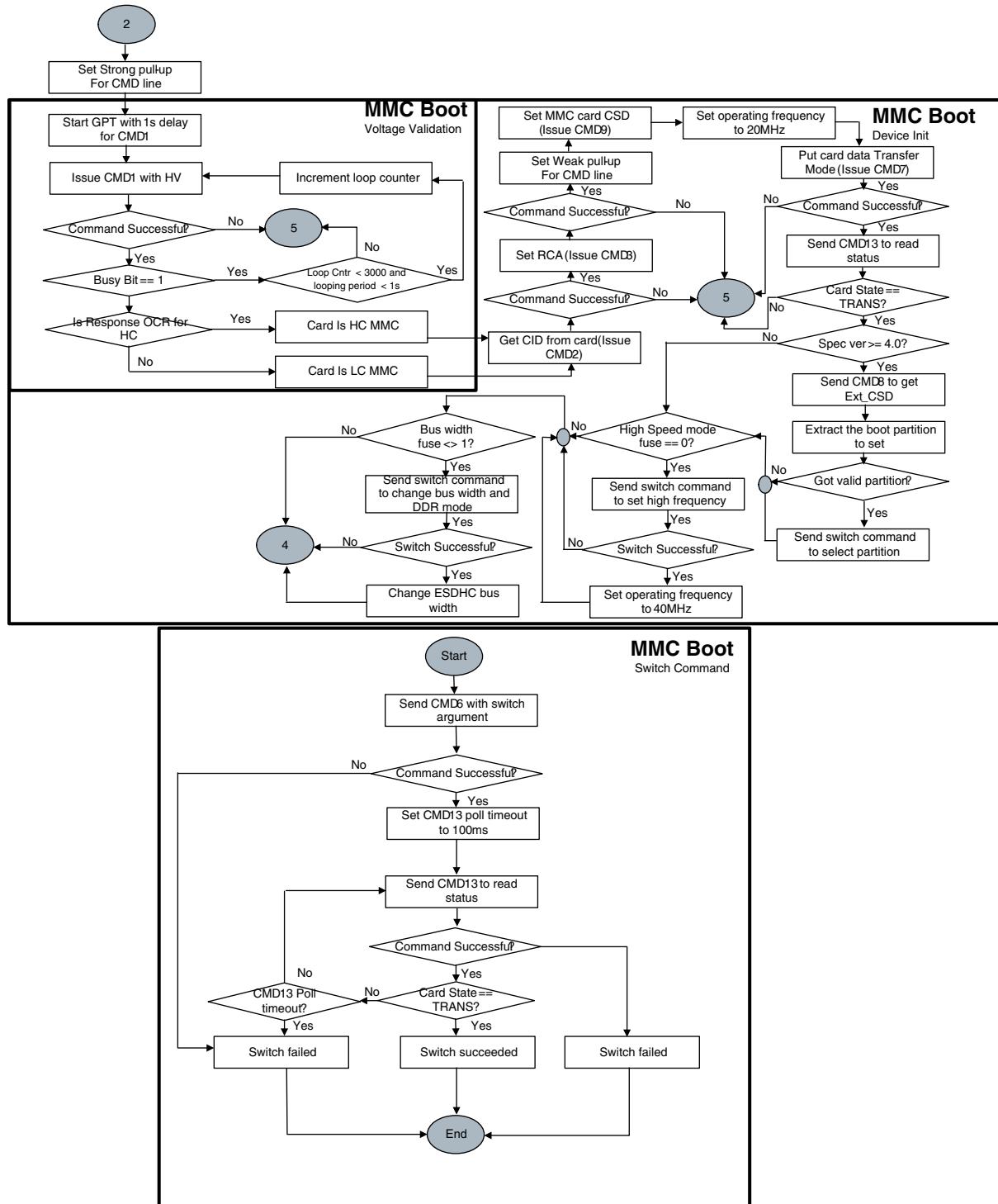


Figure 4-12. Expansion Device (MMC) Boot Flow (2 of 6)

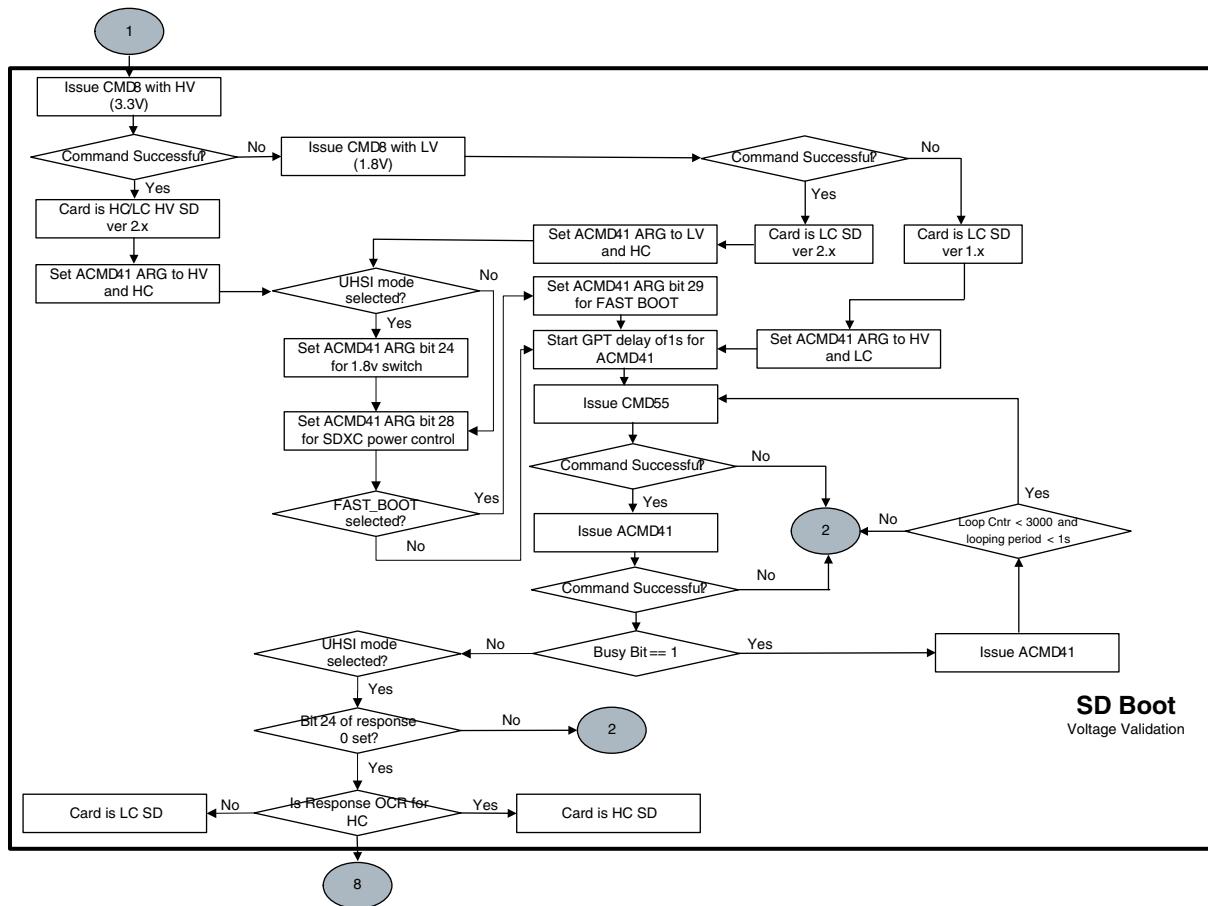


Figure 4-13. Expansion Device (SD/eSD/SDXC) Boot Flow (3 of 6) Part 1

## Boot Devices (Internal Boot)

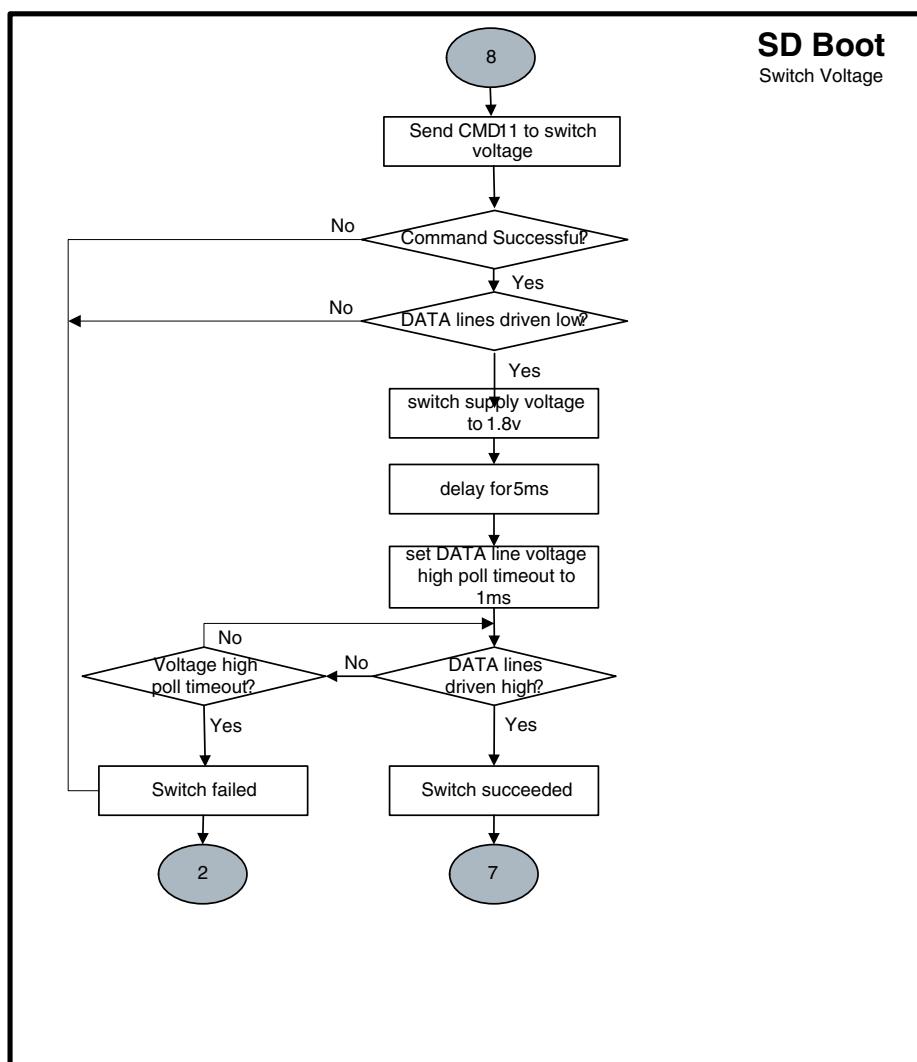
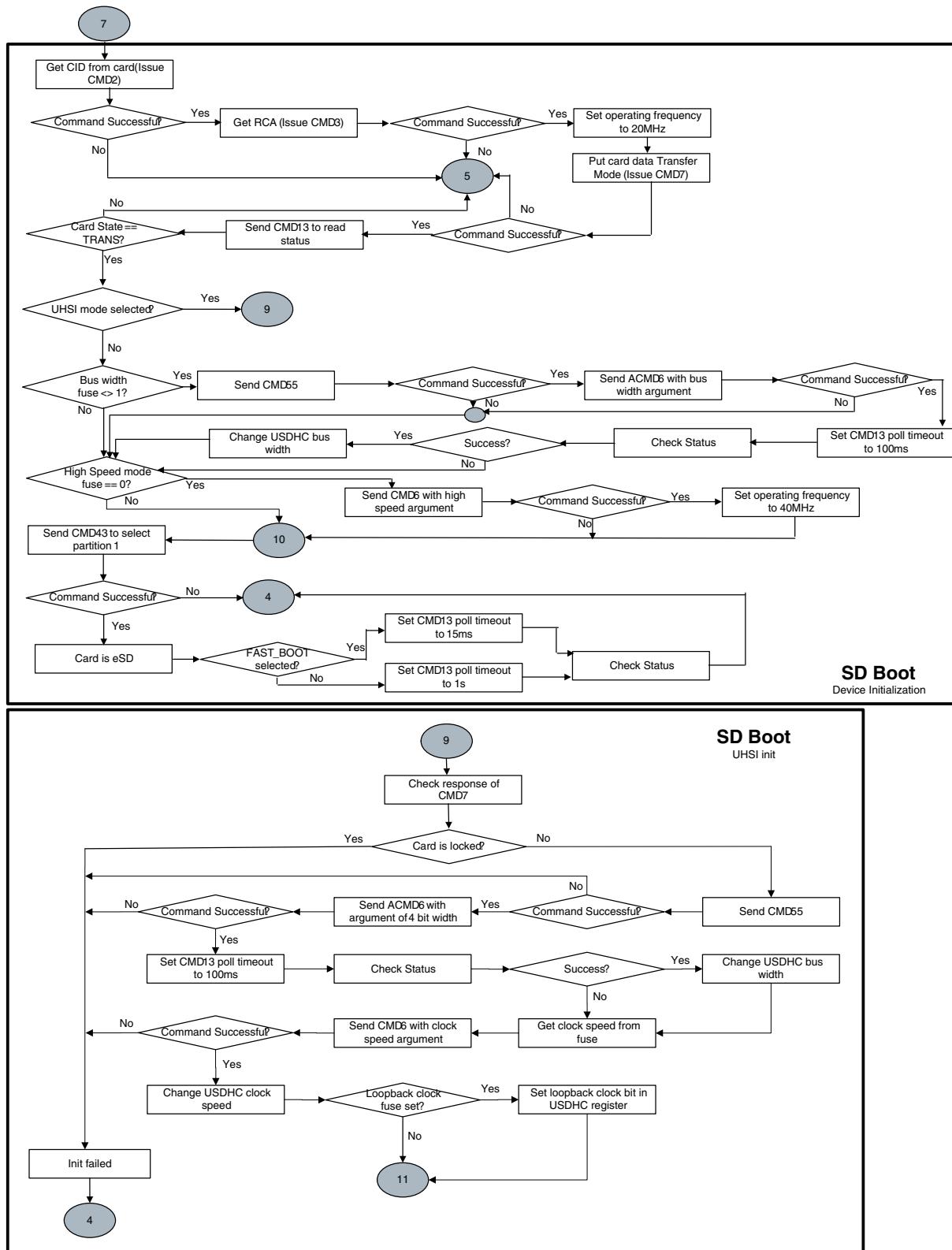
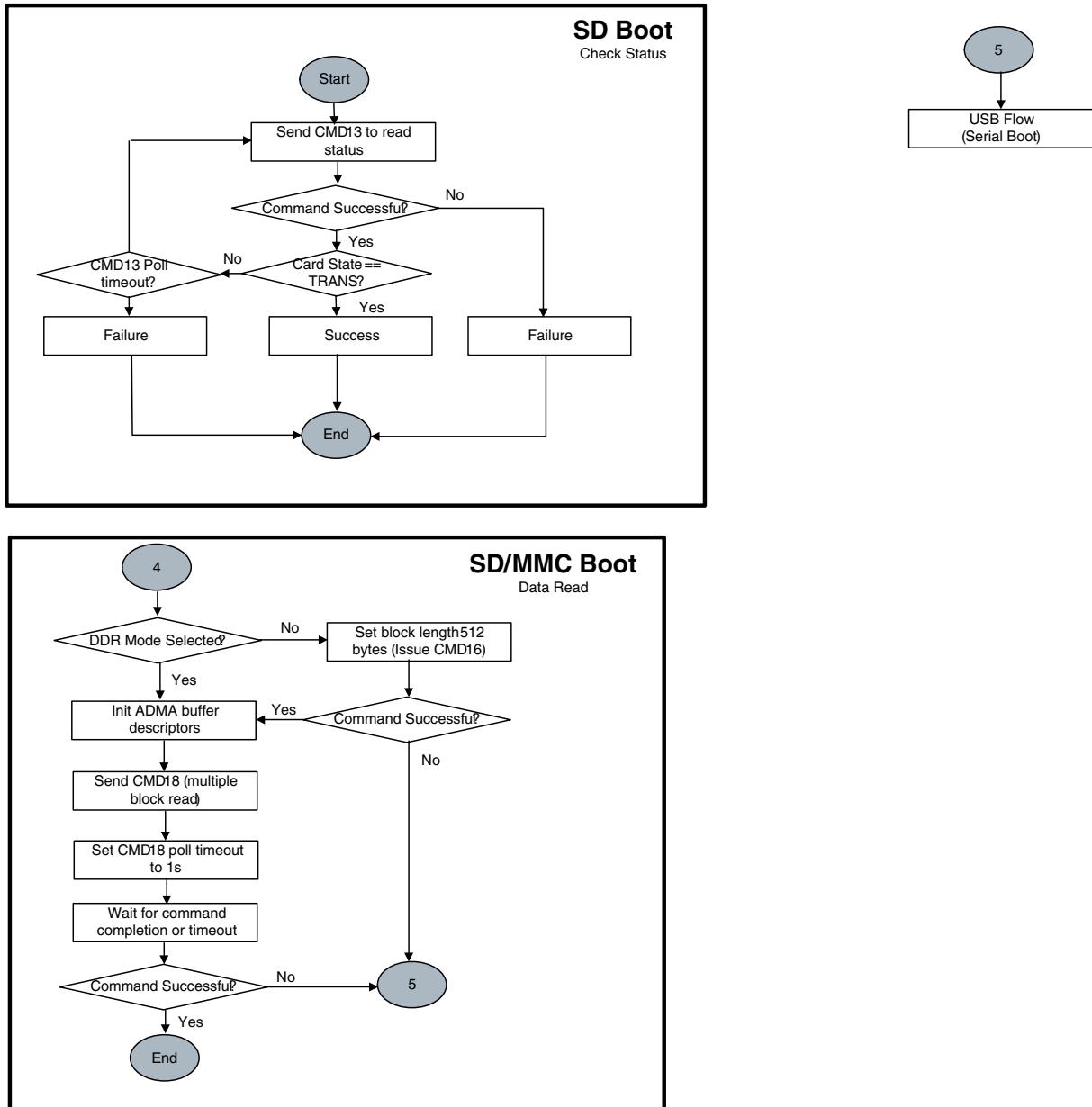


Figure 4-14. Expansion Device (SD/eSD/SDXC) Boot Flow (3 of 6) Part 2

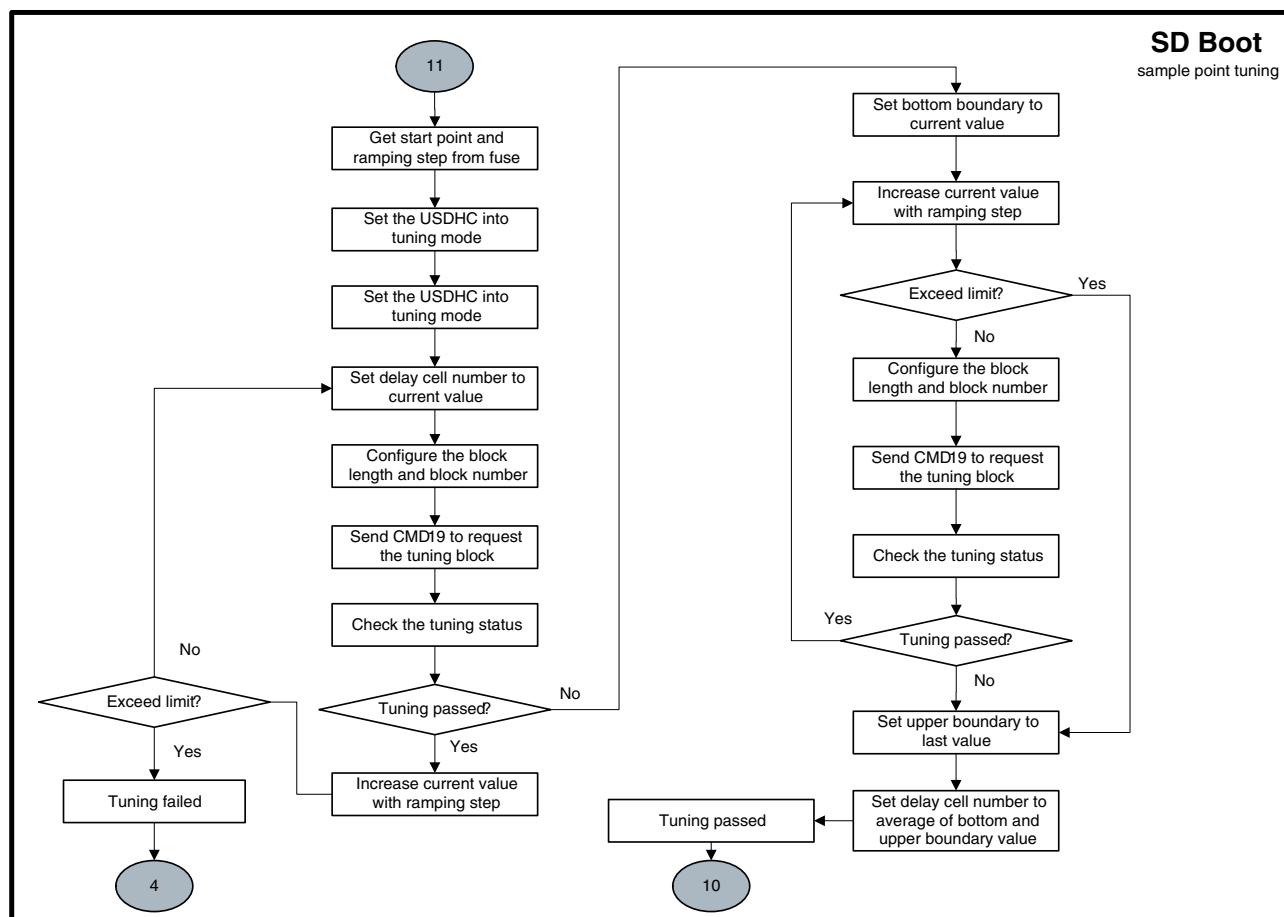
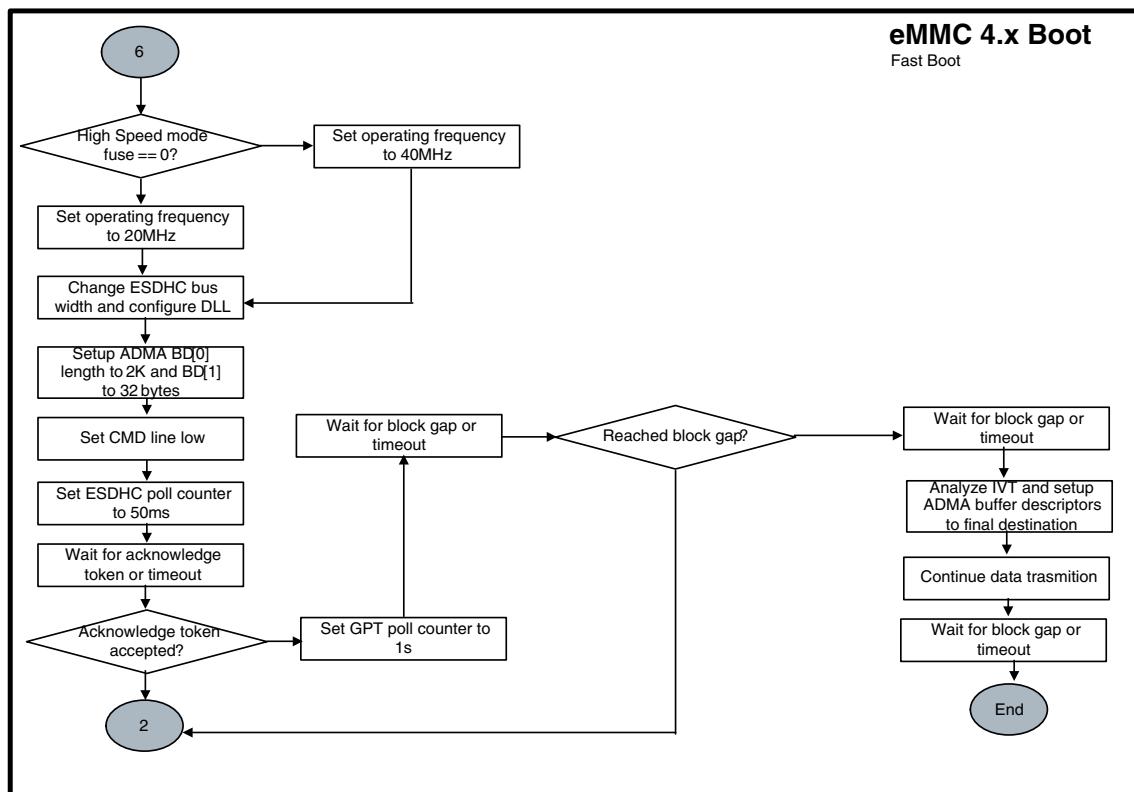


**Figure 4-15. Expansion Device (MMCSD/eSD/SDXC) Boot Flow (4 of 6)**  
**Security Reference Manual for i.MX 6Dual, 6Quad, 6Solo, and 6DualLite Families of Applications**  
**Processors, Rev. D, 11/2012**

## Boot Devices (Internal Boot)



**Figure 4-16. Expansion Device (SD/eSD) Boot Flow (5 of 6)**



**Figure 4-17. Expansion Device Boot Flow (6 of 6)**  
Security Reference Manual for i.MX 6Dual, 6Quad, 6Solo, and 6DualLite Families of Applications Processors, Rev. D, 11/2012

### 4.5.3.3 SD, eSD and SDXC

After the normal boot mode initialization begins, the SD/eSD/SDXC frequency is set to 347.22 kHz. During the identification phase, SD/eSD/SDXC card voltage validation is performed. During voltage validation, boot code first checks with high voltage settings; if that fails, it checks with low voltage settings.

The capacity of the card is also checked. Boot code supports high capacity and low capacity SD/eSD/SDXC cards after voltage validation card initialization is done.

During card initialization, the ROM boot code attempts to set the boot partition for all SD, eSD, and SDXC devices. If this fails, the boot code assumes the card is a normal SD card or SDXC card. If it does not fail, the boot code assumes it is an eSD card. After the initialization phase is over, boot code switches to a higher frequency (25 MHz in Normal Speed mode or 50 MHz in High Speed Mode). ROM also supports FAST\_BOOT mode booting from eSD card. This mode can be selected by BOOT\_CFG1[4] (Fast Boot) fuse described in [Table 4-15](#).

For UHSI cards, clock speed fuses can be set to SDR50 or SDR104 on USDHC3 and USDHC4 ports. This will enable the voltage switch process to set the signaling voltage to 1.8V during voltage validation. The bus width is fixed at 4 bits wide and a sampling point tuning process is needed to calibrate the number of delay cells. If SD Loopback Clock eFuse is set, the feedback clock will come directly from the loopback SD clock, instead of the card clock (by default). The SD clock speed can be selected by BOOT\_CFG1[3:2], and the SD Loopback Clock is selected by BOOT\_CFG1[0].

UHSI calibration start value (MMC\_DLL\_DLY[6:0]) and step value (BOOT\_CFG2[7:5]) can be set to optimize the sample point tuning process.

If SD Power Cycle Enable eFuse is 1, ROM will set SD\_RST pad low, wait 5ms and then set SD\_RST pad high. If SD\_RST pad is connected to SD power supply enable logic on board, it enables power cycle of SD card. This may be crucial in case when SD logic is in 1.8V states and must be reset to 3.3V states.

### 4.5.3.4 IOMUX Configuration for SD/MMC

**Table 4-18. SD/MMC IOMUX Pin Configuration**

Signal	USDHC-1	USDHC-2	USDHC-3	USDHC-4
<b>CLK</b>	SD1_CLK.alt0	SD2_CLK.alt0	SD3_CLK.alt0	SD4_CLK.alt0
<b>CMD</b>	SD1_CMD.alt0	SD2_CMD.alt0	SD3_CMD.alt0	SD4_CMD.alt0
<b>DAT0</b>	SD1_DAT0.alt0	SD2_DAT0.alt0	SD3_DAT0.alt0	SD4_DAT0.alt0

*Table continues on the next page...*

**Table 4-18. SD/MMC IOMUX Pin Configuration (continued)**

Signal	USDHC-1	USDHC-2	USDHC-3	USDHC-4
<b>DAT1</b>	SD1_DAT1.alt0	SD2_DAT1.alt0	SD3_DAT1.alt0	SD4_DAT1.alt0
<b>DAT2</b>	SD1_DAT2.alt0	SD2_DAT2.alt0	SD3_DAT2.alt0	SD4_DAT2.alt0
<b>DAT3</b>	SD1_DAT3.alt0	SD2_DAT3.alt0	SD3_DAT3.alt0	SD4_DAT3.alt0
<b>DAT4</b>	NANDF_D0.alt1	NANDF_D5.alt1	SD3_DAT4.alt0	SD4_DAT4.alt0
<b>DAT5</b>	NANDF_D1.alt1	NANDF_D6.alt1	SD3_DAT5.alt0	SD4_DAT5.alt0
<b>DAT6</b>	NANDF_D2.alt1	NANDF_D7.alt1	SD3_DAT6.alt0	SD4_DAT6.alt0
<b>DAT7</b>	NANDF_D3.alt1	NANDF_D8.alt1	SD3_DAT7.alt0	SD4_DAT7.alt0
<b>VSELECT</b>			GPIO_18.alt2	NANDF_CS1.alt1
<b>RESET</b> <sup>1</sup>			SD3_RESET.alt0	NANDF_CS1.alt1

1. Active low

#### 4.5.3.5 Redundant Boot Support for Expansion Device

ROM supports redundant boot for expansion device. Primary or Secondary image is selected depending on PERSIST\_SECONDARY\_BOOT setting (see [Table 4-7](#)).

If PERSIST\_SECONDARY\_BOOT is 0, the boot ROM uses address 0x0 for primary image.

If PERSIST\_SECONDARY\_BOOT is 1, the boot ROM will read secondary image table from address 0x200 on boot media and will use address specified in the table.

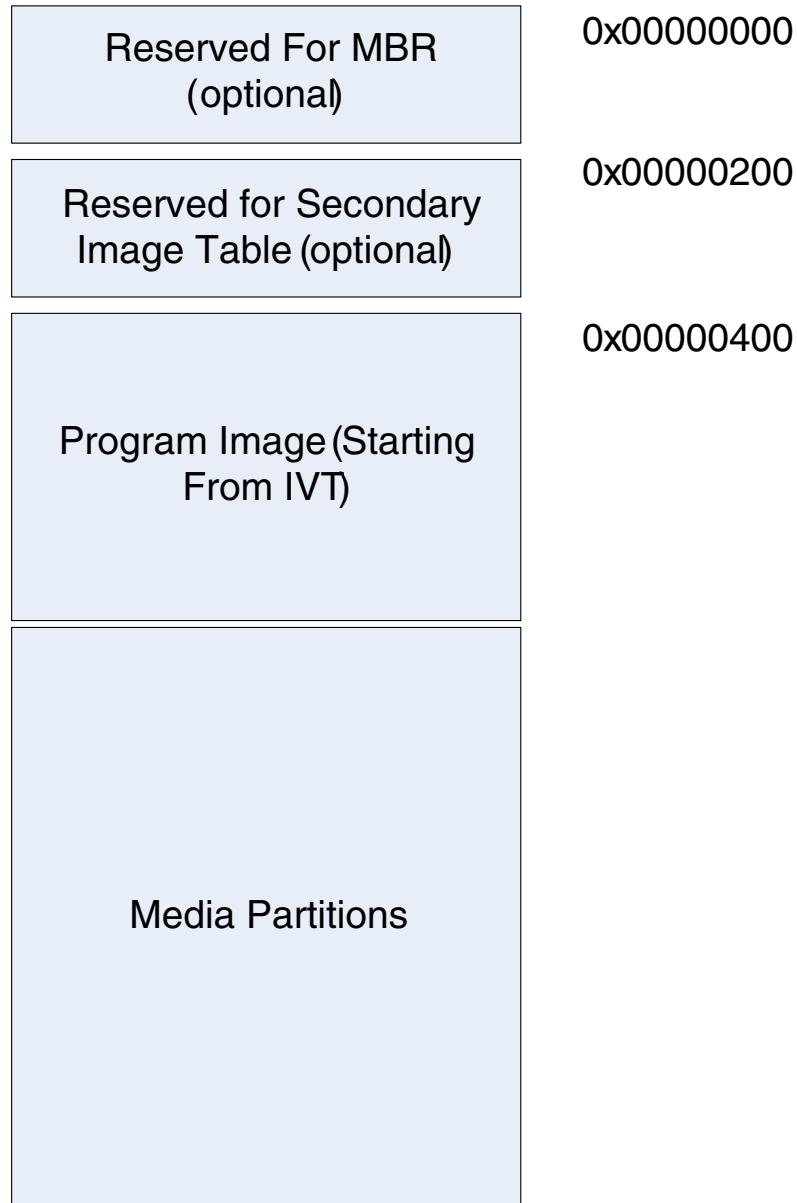
**Table 4-19. Secondary Image Table Format**

Reserved (chipNum)
Reserved (driveType)
tag
firstSectorNumber
Reserved (sectorCount)

Where:

- tag: used as indication of valid secondary image table. Must be 0x00112233.
- firstSectorNumber is the first 512B sector number of the secondary image.

For secondary image support, the primary image must reserve space for secondary image table. See the figure below for typical structures layout on expansion device.

**Figure 4-18. Expansion Device Structures Layout**

For Closed mode, if there are failures during primary image authentication, the boot ROM will turn on PERSIST\_SECONDARY\_BOOT bit (see [Table 4-7](#)) and perform software reset. (After software reset, secondary image will be used.)

#### 4.5.4 Hard Disk and SSD

The chip supports boot from Hard Disk and SSD devices using SATA interface.

#### 4.5.4.1 Hard Disk and SSD eFUSE Configuration

The boot ROM code determines the type of device using the following parameters, either provided by eFUSE settings or sampled on the I/O pins, during boot.

**Table 4-20. HDD eFUSE Descriptions**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[7:4]	OEM	Boot Device Selection	Yes	0000	0010 - Boot from Hard Disk
BOOT_CFG2[4]	OEM	Tx Spread Spectrum	Yes	0	0 - Disabled 1 - Enabled
BOOT_CFG2[3]	OEM	Rx Spread Spectrum	Yes	0	0 - Enabled 1 - Disabled
BOOT_CFG2[2]	OEM	SATA Speed	Yes	0	0 - Gen2 (3.0Gbps) 1 - Gen1 (1.5Gbps)
BOOT_CFG2[1:0]	OEM	SATA Type	Yes	00	00 - i (internal electrical specifications with cable length up to 1m, see Serial ATA specification) 01 - m (electrical specifications used in Short Backplane Application and External Desktop Application, see Serial ATA specification) 10 - x (external cabled applications or Long Backplane Applications, see Serial ATA specification) 11 - Reserved

The boot ROM will send IDENTIFY command to hard disk during initialization. When identification block is received, boot ROM assumes that the device is ready. The boot ROM sends a separate command for each sector of 512 bytes in PIO mode.

The boot ROM will copy 4KB of data from Hard Disk or SSD device to internal RAM. After checking the Image Vector Table header value (0xD1) from Program Image, the ROM code performs a DCD check. After successful DCD extraction, the ROM code extracts the destination pointer and length of image to be copied to RAM device from the Boot Data Structure, where code execution occurs.

#### NOTE

The Initial 4 KB of Program Image must contain the IVT, DCD and the Boot Data structures.

#### 4.5.4.2 IOMUX and Timing Configuration for SATA

The interface signals of the SATA PHY are not configured in the IOMUX. The SATA PHY interface uses dedicated contacts on the IC. See the Chip data sheet for details.

ROM reads the TX Spread Spectrum, RX Spread Spectrum, Speed and Type of SATA and configures timing parameters via the IOMUX GPR register.

#### 4.5.4.3 Redundant Boot Support for Hard Disk and SSD

ROM supports redundant boot for hard disk and SSD. Primary or Secondary image is selected depending on PERSIST\_SECONDARY\_BOOT setting (see [Table 4-7](#)).

If PERSIST\_SECONDARY\_BOOT is 0, the boot ROM uses address 0x0 for primary image.

If PERSIST\_SECONDARY\_BOOT is 1, the boot ROM will read secondary image table from address 0x200 on boot media and will use address specified in the table.

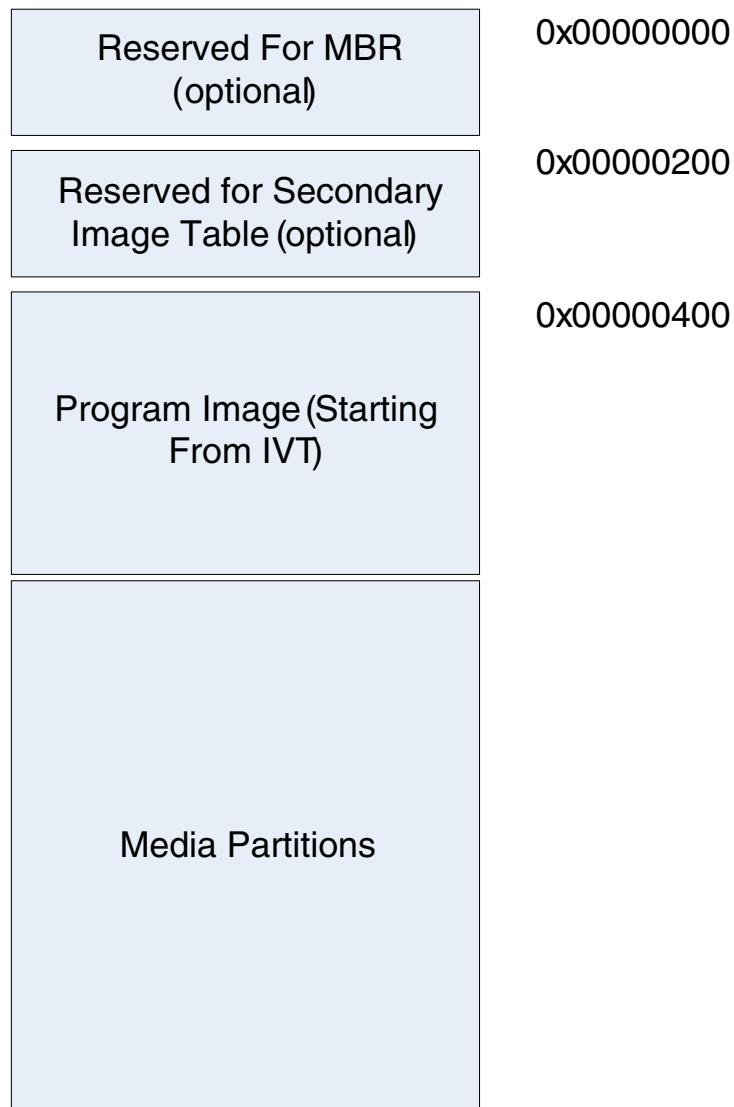
**Table 4-21. Secondary Image Table Format**

Reserved (chipNum)
Reserved (driveType)
tag
firstSectorNumber
Reserved (sectorCount)

Where:

- tag: used as indication of valid secondary image table. Must be 0x00112233.
- firstSectorNumber is the first 512B sector number of the secondary image.

For secondary image support, the primary image must reserve space for secondary image table. See the figure below for typical structures layout on expansion device.



**Figure 4-19. Hard Disk Structures Layout**

For Closed mode, if there are failures during primary image authentication, the boot ROM will turn on PERSIST\_SECONDARY\_BOOT bit (see [Table 4-7](#)) and perform software reset. (After software reset secondary image will be used).

#### 4.5.5 Serial ROM through SPI and I2C

The chip supports boot from serial memory devices, such as EEPROM and Serial Flash using the SPI.

## Boot Devices (Internal Boot)

The following ports are available for serial boot: ECSPI ( ECSPI-1, ECSPI-2, ECSPI-3, ECSPI-4, ECSPI-5), and I2C Controller ( I2C-1, I2C-2 and I2C-3) interfaces.

### 4.5.5.1 Serial ROM eFUSE Configuration

The boot ROM code determines the type of device using the following parameters, either provided by eFUSE settings or sampled on the I/O pins, during boot.

See the table below for details:

**Table 4-22. Serial ROM Boot eFUSE Descriptions**

Fuse	Config	Definition	GPIO	Shipped Value	Settings
BOOT_CFG1[7:4]	OEM	Boot Device Selection	Yes	0000	0011 - Boot from Serial ROM
BOOT_CFG4[5:4]	OEM	CS select (SPI only)	Yes	00	00 - CS#0 01 - CS#1 10 - CS#2 11 - CS#3
BOOT_CFG4[3]	OEM	SPI Addressing (SPI only)	Yes	0	0 - 2-bytes (16-bit) 1 - 3-bytes (24-bit)
BOOT_CFG4[2:0]	OEM	Port Select	Yes	00	000 - ECSPI-1 001 - ECSPI-2 010 - ECSPI-3 011 - ECSPI-4 100 - ECSPI-5 101 - I2C-1 110 - I2C-2 111 - I2C-3

The ECPSI-1/ECPSI-2/ECPSI-3/ECPSI-4/ECPSI-5 block can be used as boot device using ECSPI interface for serial ROM boot. The SPI interface is configured to operate at 15MHz for 3-byte addressing device and 3.75MHz for 2-byte addressing devices.

The I2C-1/I2C-2/I2C-3 block can be used as boot device using I2C interface, for serial ROM boot. The I2C interface is configured to operate at 343.75 Kbps.

The boot ROM will copy 4Kbyte of data from Serial ROM device to internal RAM. After checking the Image Vector Table header value (0xD1) from Program Image, the ROM code performs a DCD check. After successful DCD extraction, the ROM code extracts from Boot Data Structure the destination pointer and length of image to be copied to RAM device from where code execution occurs.

**NOTE**

The Initial 4K of Program Image must contain the IVT, DCD and the Boot Data structures.

#### 4.5.5.2 I2C Boot

The boot flow when booting from an I2C device is shown in [Figure 4-20](#).

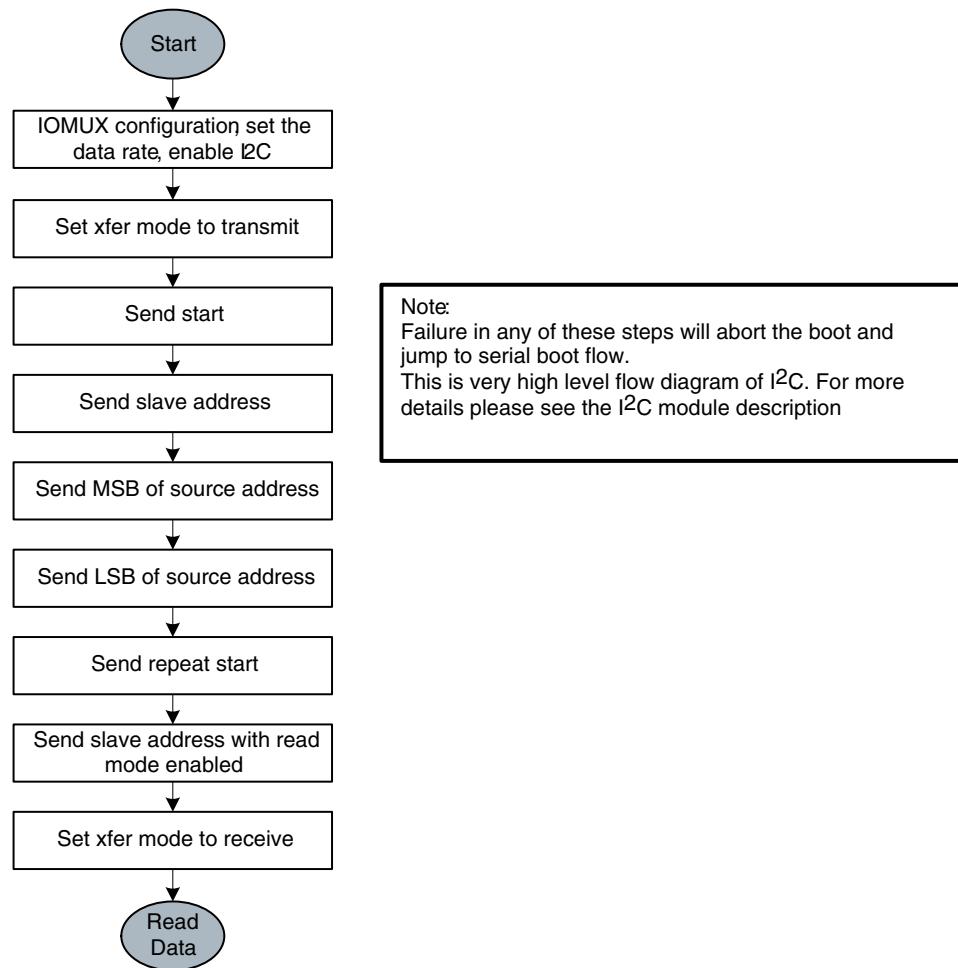
The boot ROM code reads the fuses BOOT\_CFG1[7:4] (Boot Device Selection) and BOOT\_CFG1[7:4] (Port select) to detect EEPROM device type. The ROM program copies 4K data from the EEPROM device to internal RAM. The boot ROM code next copies the initial 4Kbyte of data as well as rest of image directly to application destination extracted from application image.

The chip uses the Device Select Code/Device Address in the table below to boot from an EEPROM.

**Table 4-23. EEPROM via I2C Device Select Code**

Bits	Device Type Identifier				Chip Enable Address			R/W
	7	6	5	4	3	2	1	
Device Select Code	1	0	1	0	0	0	0	R/W

## Boot Devices (Internal Boot)



**Figure 4-20. I<sup>2</sup>C Flow Chart**

### 4.5.5.2.1 I<sup>2</sup>C IOMUX Pin Configuration

The contacts assigned to the signals used by the three I<sup>2</sup>C blocks is shown in the table below.

**Table 4-24. I<sup>2</sup>C IOMUX Pin Configuration**

Signal	I <sup>2</sup> C-1	I <sup>2</sup> C-2	I <sup>2</sup> c-3
SDA	EIM_D28.alt1	EIM_D16.alt6	EIM_D18.alt6
SCL	EIM_D21.alt6	EIM_EB2.alt6	EIM_D17.alt6

### 4.5.5.3 ECSPI Boot

The Enhanced Configurable SPI (ECSPI) interface is configured in master mode and the EEPROM device is connected to ECSPI interface as a slave.

The boot ROM code copies 4 KB data from EEPROM device to the internal RAM. If DCD verification is successful, the ROM code copies the initial 4 KB data, as well as the rest of the image extracted from application image, directly to the application destination. The ECSPI can read data from EEPROM using 2 or 3 byte addressing. Its burst length is 32 bytes.

#### NOTE

The Serial ROM Chip Select Number is determined by  
BOOT\_CFG4[5:4] (Chip Select) fuse.

When using the SPI as boot device, the Chip supports booting from both Serial EEPROM and Serial Flash devices. The boot code determines which device is being used by reading the appropriate eFUSE/I/O values at boot (see [Table 4-22](#) for details).

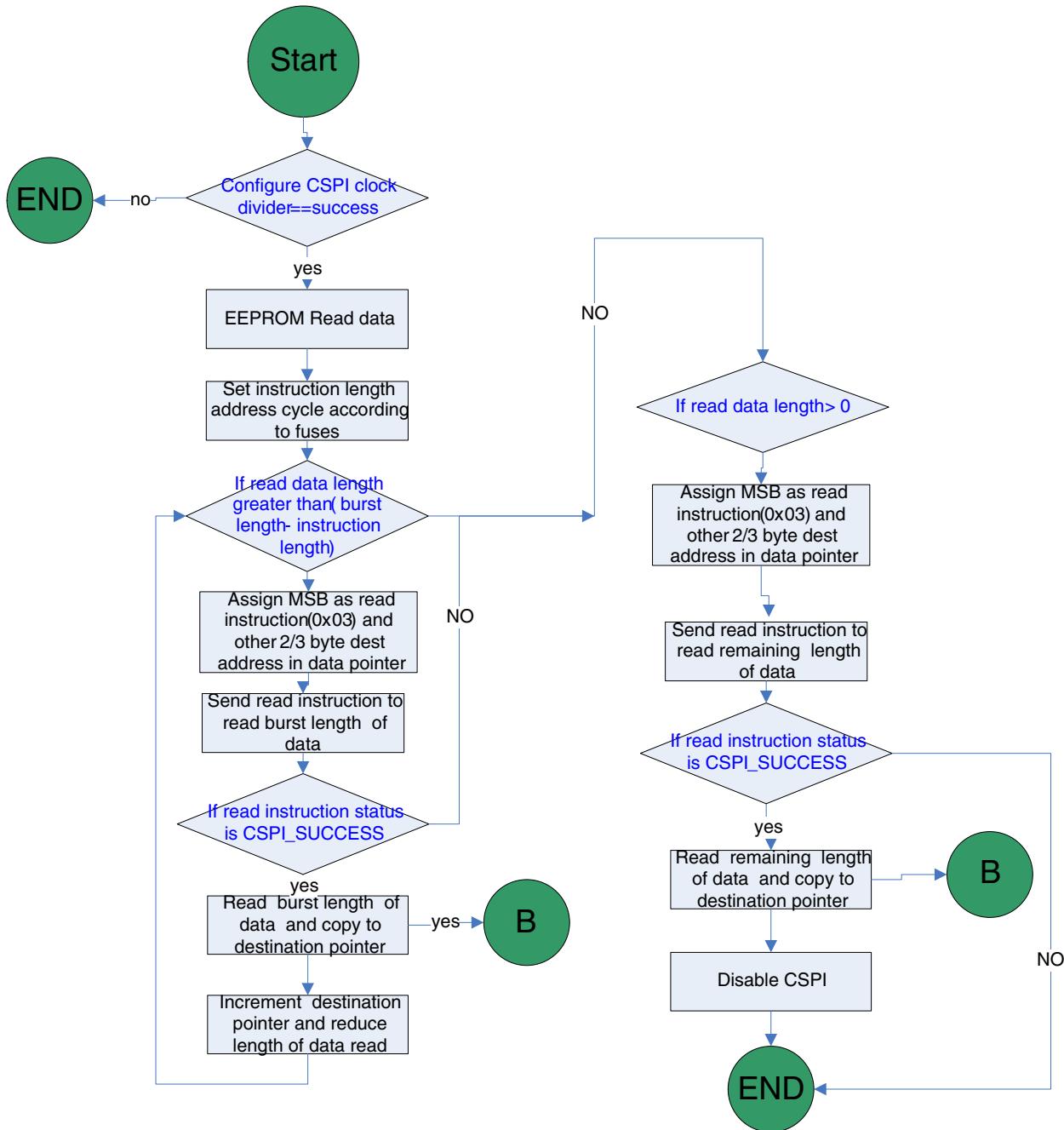


Figure 4-21. CSPI Flow chart

#### 4.5.5.3.1 ECSPI IOMUX Pin Configuration

The contacts assigned to the signals used by the three CSPI blocks is shown in the table below.

**Table 4-25. SPI IOMUX Pin Configuration**

Signal	ECSPI-1	ECSPI-2	ECSPI-3	ECSPI4	ECSPI-5
<b>MISO</b>	EIM_D17.alt1	CSI0_DAT10.alt2	DISP0_DAT2.alt2	EIM_D22.alt1	SD1_DAT0.alt1N/A
<b>MOSI</b>	EIM_D18.alt1	CSI0_DAT9.alt2	DISP0_DAT1.alt2	EIM_D28.alt2	SD1_CMD.alt1N/A
<b>RDY</b>	N/A	N/A	N/A	N/A	N/A
<b>SCLK</b>	EIM_D16.alt1	CSI0_DAT8.alt2	DISP0_DAT0.alt2	EIM_D21.alt1	SD1_CLK.alt1N/A
<b>SS0</b>	EIM_EB2.alt1	CSI0_DAT11.alt2	DISP0_DAT3.alt2	EIM_D20.alt1	SD1_DAT1.alt1N/A
<b>SS1</b>	EIM_D19.alt1	EIM_LBA.alt1	DISP0_DAT4.alt2	EIM_A25.alt1	SD1_DAT2.alt1N/A
<b>SS2</b>	EIM_D24.alt1	EIM_D24.alt4	DISP0_DAT5.alt2	EIM_D24.alt1	SD1_DAT3.alt1N/A
<b>SS3</b>	EIM_D25.alt1	EIM_D25.alt4	DISP0_DAT6.alt2	EIM_D25.alt1	SD2_DAT3.alt1N/A

## 4.6 Program image

This section describes the data structures that are required to be included in a user's program image. A program image consists of:

- Image vector table—A list of pointers located at a fixed address that the ROM examines to determine where other components of the program image are located
- Boot data—A table indicating the program image location, program image size in bytes, and the plugin flag
- Device configuration data—IC configuration data
- Data used by HAB—CSF command data, signatures and certificates (optional for non-secure boot). CSF commands are the instructions HAB uses to determine which keys to install and which memory areas correspond to the digital signatures.
- User code and data

### 4.6.1 Image Vector Table and Boot Data

The Image Vector Table (IVT) is the data structure that the ROM reads from the boot device supplying the program image containing the required data components to perform a successful boot.

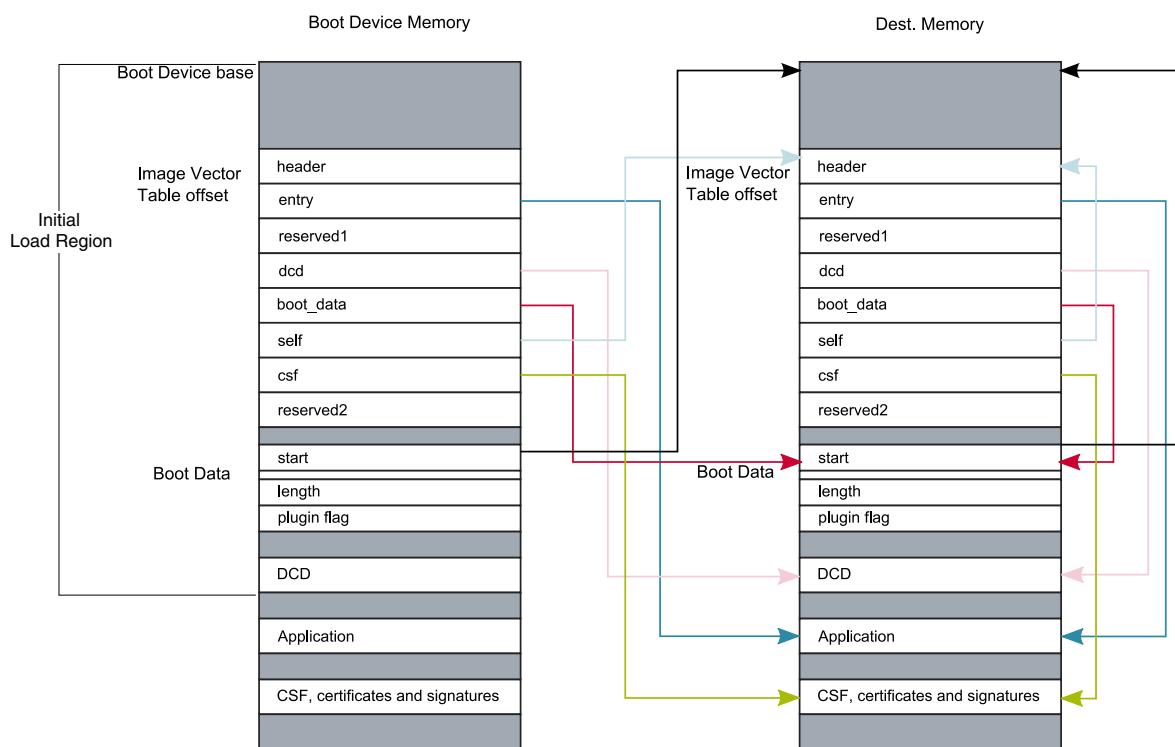
The IVT includes the program image entry point, a pointer to Device Configuration Data (DCD) and other pointers used by the ROM during the boot process. The ROM locates the IVT at a fixed address that is determined by the boot device connected to the Chip. The IVT offset from the base address and initial load region size for each boot device

## Program image

type is defined in the table below. The location of the IVT is the only fixed requirement by the ROM. The remainder or the image memory map is flexible and is determined by the contents of the IVT.

**Table 4-26. Image Vector Table Offset and Initial Load Region Size**

Boot Device Type	Image Vector Table Offset	Initial Load Region Size
NOR	4 Kbyte = 0x1000 bytes	Entire Image Size
NAND	1 Kbyte = 0x400 bytes	4 Kbyte
OneNAND	256 bytes = 0x100 bytes	1 Kbyte
SD/MMC/eSD/eMMC/SDXC	1 Kbyte = 0x400 bytes	4 Kbyte
I2C/SPI EEPROM	1 Kbyte = 0x400 bytes	4 Kbyte
SATA	1 Kbyte = 0x400 bytes	4 Kbyte



**Figure 4-22. Image Vector Table**

### 4.6.1.1 Image Vector Table Structure

The IVT has the following format where each entry is a 32 bit word:

**Table 4-27. IVT Format**

header
entry: Absolute address of the first instruction to execute from the image
reserved1: Reserved and should be zero
dcd: Absolute address of the image DCD. The DCD is optional so this field may be set to NULL if no DCD is required. See <a href="#">Device Configuration Data (DCD)</a> for further details on DCD.
boot data: Absolute address of the Boot Data
self: Absolute address of the IVT. Used internally by the ROM
csf: Absolute address of Command Sequence File (CSF) used by the HAB library. See <a href="#">High Assurance Boot (HAB)</a> for details on secure boot using HAB. This field must be set to NULL when not performing a secure boot
reserved2: Reserved and should be zero

The IVT header has the following format:

**Table 4-28. IVT Header Format**

Tag	Length	version
-----	--------	---------

where:

Tag: A single byte field set to 0xD1

Length: a two byte field in big endian format containing the overall length of the IVT, in bytes, including the header. (the length is fixed and must have a value of 32 bytes)

Version: A single byte field set to 0x40 or 0x41

#### 4.6.1.2 Boot Data Structure

The Boot Data must follow the format defined in the table below, each entry is a 32-bit word.

**Table 4-29. Boot Data Format**

start	Absolute address of the image
length	Size of the program image
plugin	Plugin flag (see <a href="#">Plugin Image</a> )

#### 4.6.2 Device Configuration Data (DCD)

Upon reset, the Chip uses the default register values for all peripherals in the system. However, these settings typically are not ideal for achieving optimal system performance and there are even some peripherals that must be configured before they can be used.

## Program image

The DCD is configuration information contained in a Program Image, external to the ROM, that the ROM interprets to configure various peripherals on the Chip.

For example, the EIM default settings allow the core to interface to a NOR flash device immediately out of reset. This allows the Chip to interface with any NOR flash device, but has the cost of slow performance. Additionally, some components such as DDR require some sequence of register programming as part of configuration before it is ready to be used. The DCD feature can be used to program the EIM registers and MMDC registers to the optimal settings.

The ROM determines the location of the DCD table based on information located in the Image Vector Table (IVT). See [Image Vector Table and Boot Data](#) for more details. The DCD table shown below is a big endian byte array of the allowable DCD commands. The maximum size of the DCD limited to 1768 bytes.

**Table 4-30. DCD Data format**

Header
[CMD]
[CMD]
...

The DCD header is 4 bytes with the following format:

**Table 4-31. DCD Header**

Tag	Length	Version
-----	--------	---------

where:

Tag: A single byte field set to 0xD2  
Length: a two byte field in big endian format containing the overall length of the DCD, in bytes, including the header  
Version: A single byte field set to 0x41

### 4.6.2.1 Write Data Command

The Write Data Command is used to write a list of given 1-, 2- or 4-byte values or bitmasks to a corresponding list of target addresses.

The format of Write Data Command, again a big endian byte array, is shown in the table below.

**Table 4-32. Write Data Command Format**

Tag	Length	Parameter
-----	--------	-----------

*Table continues on the next page...*

**Table 4-32. Write Data Command Format (continued)**

Address
Value/Mask
[Address]
[Value/Mask]
...
[Address]
[Value/Mask]

where:

Tag: A single byte field set to 0xCC

Length: A two byte field in big endian format containing the length of the Write Data Command, in bytes, including the header

Address: target address to which data should be written

Value/Mask: data value or bitmask to be written to preceding address

The Parameter field is a single byte divided into bitfields as follows:

**Table 4-33. Write Data Command Parameter field**

7	6	5	4	3	2	1	0
flags						bytes	

where

bytes: width of target locations in bytes. Either 1, 2 or 4

flags: control flags for command behavior.

Data Mask = bit 3: if set, only specific bits may be overwritten at target address  
(otherwise all bits may be overwritten)

Data Set = bit 4: if set, bits at the target address overwritten with this flag  
(otherwise it is ignored)

One or more target address and value/bitmask pairs can be specified. The same bytes and flags parameters apply to all locations in the command.

When successful, this command writes to each target address in accordance with the flags as follows:

**Table 4-34. Interpretation of Write Data Command Flags**

"Mask"	"Set"	Action	Interpretation
0	0	*address = val_msk	Write value
0	1	*address = val_msk	Write value
1	0	*address &= ~val_msk	Clear bitmask
1	1	*address  = val_msk	Set bitmask

**NOTE**

If any of the target addresses does not have the same alignment as the data width indicated in the parameter field, none of the values are written.

If any of the values is larger or any of the bitmasks is wider than permitted by the data width indicated in the parameter field, none of the values are written.

If any of the target addresses do not lie within an allowed region, none of the values are written. The list of allowable blocks and target addresses for the Chip are given below.

**Table 4-35. Valid DCD Address Ranges**

<b>Address range</b>	<b>Start address</b>	<b>Last Address</b>
IOMUX Control (IOMUXC) registers	0x020E0000	0x020E3FFF
CCM register set	0x020C4000	0x020C7FFF
ANADIG registers	0x020C8000	0x020C8FFF
MMDC register set	0x021B0000	0x021B7FFF
IRAM Free Space	0x00907000	0x00937FF0
EIM	0x08000000	0xFFFFFFF
DDR	0x10000000	0xFFFFFFFF

**4.6.2.2 Check Data Command**

The Check Data Command is used to test for a given -1, 2- or 4-byte bitmasks from a source address.

The Check Data Command is a big endian byte array with format shown in the table below.

**Table 4-36. Check Data Command Format**

Tag	Length	Parameter
	Address	
	Mask	
	[Count]	

where:

Tag: A single byte field set to 0xCF

Length: A two byte field in big endian format containing the length of the Check Data Command, in bytes, including the header

Address: source address to test

Mask: bit mask to test

Count: optional poll count. If count is not specified this command will poll indefinitely until the exit condition is met. If count = 0, this command behaves as for NOP.

The Parameter field is a single byte divided into bitfields as follows:

**Table 4-37. Check Data Command Parameter field**

7	6	5	4	3	2	1	0
flags				bytes			

where

bytes: width of target locations in bytes. Either 1, 2 or 4

flags: control flags for command behavior.

Data Mask = bit 3: if set, only specific bits may be overwritten at target address  
(otherwise all bits may be overwritten)

Data Set = bit 4: if set, bits at the target address overwritten with this flag  
(otherwise it is ignored)

This command polls the source address until either the exit condition is satisfied, or the poll count is reached. The exit condition is determined by the flags as follows:

**Table 4-38. Interpretation of Check Data Command Flags**

"Mask"	"Set"	Action	Interpretation
0	0	(*address & mask) == 0	All bits clear
0	1	(*address & mask) == mask	All bits set
1	0	(*address & mask)!= mask	Any bit clear
1	1	(*address & mask)!= 0	Any bit set

## NOTE

If the source address does not have the same alignment as the data width indicated in the parameter field, the value is not read.

If the bitmask is wider than permitted by the data width indicated in the parameter field, the value is not read.

### 4.6.2.3 NOP Command

This command has no effect.

## Plugin Image

The format of NOP Command is a big endian four byte array as shown in the table below.

**Table 4-39. NOP Command Format**

Tag	Length	Undefined
-----	--------	-----------

where:

Tag: A single byte field set to 0xC0  
Length: A two byte field in big endian containing the length of the NOP Command in bytes.  
Fixed to a value of 4.  
Undefined: This byte is ignored and can be set to any value.

### 4.6.2.4 Unlock Command

The Unlock Command is used to prevent specific engine features being locked when exiting ROM.

The format of Unlock Command, again a big endian byte array, is shown in the table below.

**Table 4-40. Unlock Command Format**

Tag	Length	Eng
	Value	
	Value	
	...	
	Value	

where:

Tag: A single byte field set to 0xB2  
Eng: Engine to be left unlocked  
Value: [optional] unlock values required by eng

#### NOTE

This command may not be used in DCD structure if the SEC\_CONFIG is configured as closed.

## 4.7 Plugin Image

The ROM supports a limited number of boot devices. For using other devices as boot source (for example, Ethernet, CDROM, or USB), the supported boot device must be used (typically serial ROM) for firmware with the missing boot drivers.

Additionally plugin can customize supported boot drivers. It is more flexible when doing device initialization, such as condition judging, delay assertion, applying custom settings to boot device and memory system.

In addition to standard images, the chip also supports plugin images. Plugin images return execution to the ROM whereas a standard image does not. The boot ROM detects the image type using the plugin flag of the boot data structure (see [Boot Data Structure](#)). If the plugin flag is 1, then the ROM uses the image as a plugin function. The function must initialize the boot device and copy the program image to the final location. At the end the plugin function must return with the program image parameters. (See [High level boot sequence](#) for details about boot flow).

The boot ROM authenticates the plugin image prior to running the plugin function and then authenticates the program image.

The plugin function must follow the API described below:

```
typedef unsigned char (*) plugin_download_f(void **start, size_t *bytes, UINT32
*iwt_offset)
```

#### ARGUMENTS PASSED:

- start - Image load address on exit.
- bytes - Image size on exit.
- iwt\_offset - Offset in bytes of the IVT from the image start address on exit.

#### RETURN VALUE:

- 1 - on success
- 0 - on failure

## 4.8 Serial Downloader

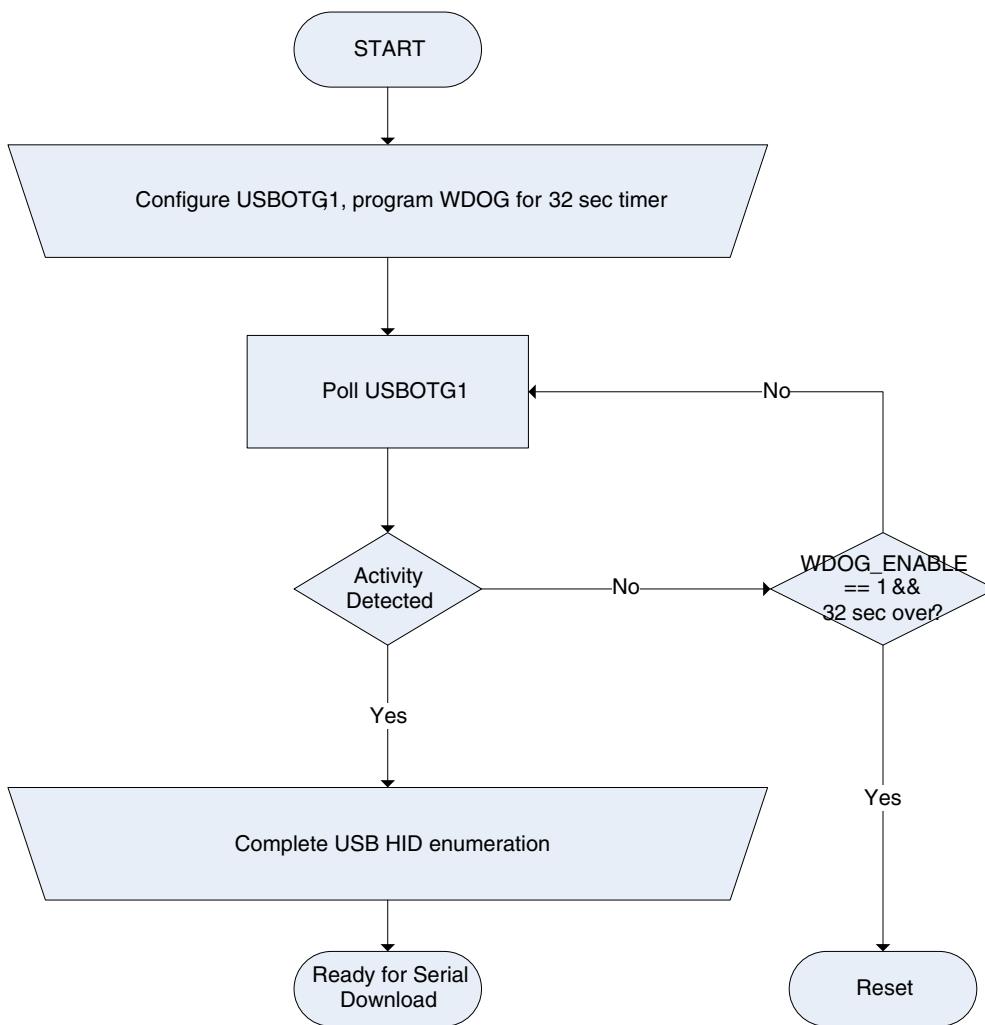
The Serial Downloader provides a means to download a Program Image to the chip over USB serial connection.

In this mode the ROM programs WDOG-1 for a 32-second time-out if WDOG\_ENABLE eFuse is 1, and then continuously polls for USB connection. If no activity is found on USB OTG1 and the watchdog timer expires, the ARM core is reset.

### NOTE

The downloaded image must continue to service the watchdog timer to avoid an undesired reset from occurring.

The USB boot flow is shown in the figure below.

**Figure 4-23. USB Boot Flow**

### 4.8.1 USB

USB support is composed of the USBOH3 (USB OTG1 core controller, compliant with the USB 2.0 specification) and the USBPHY (HS USB transceiver).

The ROM supports the USB OTG port for boot purposes. The other USB ports on the chip are not supported for boot purposes.

The USB Driver is implemented as a USB HID class. A collection of 4 HID reports are used to implement SDP protocol for data transfers as described in [Table 4-41](#).

**Table 4-41. USB HID Reports**

Report ID (first byte)	Transfer Endpoint	Direction	Length	Description
1	control OUT	Host to device	17 bytes	SDP command from host to device
2	control OUT	Host to device	Up to 1025 bytes	Data associated with report 1 SDP command
3	interrupt	Device to host	5 bytes	HAB security configuration. Device sends 0x12343412 in closed mode and 0x56787856 in open mode.
4	interrupt	Device to host	Up to 65 bytes	Data in response to SDP command in report 1

#### 4.8.1.1 USB Configuration Details

The USB OTG function device driver supports a high speed (HS for UTMI) non-stream mode with a maximal packet size of 512 B and a low-level USB OTG function.

The VID/PID and strings for USB device driver are listed in the table below.

**Table 4-42. VID/PID and Strings for USB Device Driver**

Descriptor	Value
VID	0x15A2 (Freescale vendor ID)
PID	0x00540x61
String Descriptor1 (manufacturer)	Freescale Semiconductor, Inc.
String Descriptor2 (product)	S Blank ARIKRIGEL SE Blank ARIKRIGEL NS Blank ARIKRIGEL
String Descriptor4	Freescale Flash
String Descriptor5	Freescale Flash

### 4.8.1.2 IOMUX Configuration for USB

The interface signals of the UTMI PHY are not configured in the IOMUX. The UTMI PHY interface uses dedicated contacts on the IC. See the Chip data sheet for details.

### 4.8.2 Serial Download protocol

The 16 byte SDP command from host to device is sent using HID report 1.

The table below describes 16 byte SDP command data structure:

**Table 4-43. 16 Byte SDP Command Data Structure**

BYTE Offset	Size	Name	Description
0	2	COMMAND TYPE	The following commands are supported for i.MX6 Dual/6Quad ROM: <ul style="list-style-type: none"> <li>• 0x0101 READ_REGISTER</li> <li>• 0x0202 WRITE_REGISTER</li> <li>• 0x0404 WRITE_FILE</li> <li>• 0x0505 ERROR_STATUS</li> <li>• 0xA0A DCD_WRITE</li> <li>• 0xB0B JUMP_ADDRESS</li> </ul>
2	4	ADDRESS	Only relevant for following commands: READ_REGISTER, WRITE_REGISTER, WRITE_FILE, DCD_WRITE, and JUMP_ADDRESS. For READ_REGISTER and WRITE_REGISTER commands, this field is address to a register. For WRITE_FILE and JUMP_ADDRESS commands, this field is an address to internal or external memory address.
6	1	FORMAT	Format of access, 0x8 for 8-bit access, 0x10 for 16-bit and 0x20 for 32-bit access. Only relevant for READ_REGISTER and WRITE_REGISTER commands.
7	4	DATA COUNT	Size of data to read or write. Only relevant for WRITE_FILE, READ_REGISTER, WRITE_REGISTER and DCD_WRITE commands. For WRITE_FILE and DCD_WRITE commands DATA COUNT is in byte units.
11	4	DATA	Value to write. Only relevant for WRITE_REGISTER command.
15	1	RESERVED	Reserved

#### 4.8.2.1 SDP Command

SDP commands are described in the following sections.

#### 4.8.2.1.1 READ REGISTER

The transaction for command READ\_REGISTER consists of following reports: Report1 for command, Report3 for security configuration and Report4 for response or register value.

The register to read is specified in ADDRESS field of SDP command. First device sends Report3 with security configuration followed by Report4 with bytes read at given address. If count is greater than 64 then multiple reports with report id 4 are sent until entire data requested by host is sent. The STATUS is either 0x12343412 for closed parts and 0x56787856 for open or field return parts.

Report1, Command, Host to Device:

1	Valid values for READ_REGISTER COMMAND, ADDRESS, FORMAT, DATA_COUNT
---	---

ID 16 byte SDP Command

Report3, Response, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

Report4, Response, Device to Host: first response report

4	Register Value
---	----------------

ID 4 bytes of data containing register value. If number of bytes requested is less than 4 then remaining bytes should be ignored by host.

Multiple reports of report id 4 are sent until entire data requested is sent

Report4, Response, Device to Host: Last response report

4	Register Value
---	----------------

ID 64 bytes of data containing register value. If number of bytes requested is less than 64 then remaining bytes should be ignored by host.

### 4.8.2.1.2 WRITE REGISTER

The transaction for command WRITE\_REGISTER consists of the following reports: Report1 for command, Report3 for security configuration and Report4 for write status.

Host sends Report1 with WRITE\_REGISTER command. The register to write is specified in ADDRESS field of SDP command of Report1, with FORMAT field set to data type (number of bits to write 8, 16 or 32) and value to write in DATA field of SDP command. Device writes the DATA to register address and returns WRITE\_COMPLETE code using Report4 and security configuration using Report3 to complete the transaction.

Report1, Command, Host to Device:

1	Valid values for WRITE_REGISTER COMMAND, ADDRESS, FORMAT, DATA_COUNT and DATA
---	---

ID 16 byte SDP Command

Report3, Response, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

Report4, Response, Device to Host:

4	WRITE_COMPLETE (0x128A8A12) status
---	------------------------------------

ID 64 bytes data with first 4 bytes to indicate write is completed with code 0x128A8A12. On failure device will report HAB error status.

### 4.8.2.1.3 WRITE\_FILE

The transaction for command WRITE\_FILE consists of following reports: Report1 for command-phase, Report2 for data-phase, Report3 for hab mode and Report4 to indicate data received in full.

The size of each Report2 is limited to 1024 bytes (limitation of USB HID protocol) hence multiple Report2 packets will be sent by host in data phase until entire data is transferred to device. Once entire data (DATA\_COUNT bytes) is received then device sends report 3 with hab mode and report 4 with 0x88888888, indicating file download completed.

Report1, Host to Device:

1	Valid values for WRITE_FILE COMMAND, ADDRESS, DATA_COUNT
---	--

ID 16 byte SDP Command

=====Optional Begin=====

Host sends ERROR\_STATUS command to query if HAB rejected the address

===== Optional End=====

Report2, Host to Device:

2	File data
---	-----------

ID Max 1024 bytes data per report

Report2, Host to Device:

2	File data
---	-----------

ID Max 1024 bytes data per report

Report3, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

Report4, Response, Device to Host:

4	COMPLETE (0x88888888) status
---	------------------------------

ID 64 bytes data with first 4 bytes to indicate file download has completed with code 0x88888888. On failure device will report HAB error status.

#### 4.8.2.1.4 ERROR\_STATUS

The transaction for SDP command ERROR\_STATUS consists of three reports.

Report1 is used by host to send the command; device sends global error status in 4 bytes of Report4 after returning security configuration in Report3. When device receives ERROR\_STATUS command it will return global error status that is updated for each command. This command is useful to find out if last command resulted in device error or succeeded.

Report1, Command, Host to Device:

1	ERROR_STATUS COMMAND
---	----------------------

ID 16 byte SDP Command

Report3, Response, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

Report4, Response, Device to Host:

4	4 bytes Error status
---	----------------------

ID first 4 bytes status in 64 bytes report 4

#### **4.8.2.1.5 DCD WRITE**

The SDP command DCD\_WRITE is used by host to send multiple register writes in one shot. This command is provided to speed up the process of programming register writes such as to configure external RAM device.

The command goes with Report1 from host with COMMAND TYPE set to DCD\_WRITE, ADDRESS which is used for temporary location of DCD data and DATA\_COUNT to number of bytes sent in data out phase. In data phase host sends data for number of registers using Report2. Device completes the transaction with Report3 indicating security configuration and report 4 with WRITE\_COMPLETE code 0x12828212.

Report1, Command, Host to Device:

1	DCD_WRITE COMMAND, ADDRESS, DATA_COUNT
---	--

ID 16 byte SDP Command

Report2, Data, Host to Device:

2	DCD binary data
---	-----------------

ID Max 1024 bytes per report

Report3, Response, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

Report4, Response, Device to Host:

4	WRITE_COMPLETE (0x128A8A12) status
---	------------------------------------

ID 64 bytes report with first 4 bytes to indicate write is completed with code 0x128A8A12. On failure device will report HAB error status.

See [Device Configuration Data \(DCD\)](#) for DCD format description.

#### 4.8.2.1.6 JUMP ADDRESS

The SDP command JUMP\_ADDRESS will be the last command host can send to the device, after this command device will jump to the address specified in the ADDRESS field of SDP command and start executing.

This command should typically follow after WRITE\_FILE command. The command is sent by host in command-phase of transaction using Report1, there is no data phase for this command but device send status report3 to complete the transaction. And if HAB authentication fails then it will also send report 4 with HAB error status.

Report1, Command, Host to Device:

1	JUMP_ADDRESS COMMAND, ADDRESS
---	-------------------------------

ID 16 byte SDP Command

Report3, Response, Device to Host:

3	4 bytes indicating security configuration
---	---

ID 4 bytes status

This report is sent by device only in case of an error jumping to the given address, device reports error in Report4, Response, Device to Host:

4	4 bytes HAB error status
---	--------------------------

ID 4 bytes status, 64 bytes report length

## 4.9 Recovery Devices

The Chip supports recovery devices. If primary boot device fails, boot ROM will try to boot from recovery device using one of I2C or ECSPi ports.

For enabling recovery device BOOT\_CFG4[6] fuse must be set. Additionally Serial EEPROM fuses must be set as described in [Serial ROM through SPI and I2C](#).

## 4.10 USB Low Power Boot

ROM supports USB Low Power Boot. This feature enables a device with dead or weak battery to power up and boot if the device is connected to a USB upstream port, no matter the upstream port is a USB charger or USB host/hub.

If a USB dedicated charger or host/hub charger are connected, as soon as the device is connected to the upstream port, a stable current (Max.1.5A) can be supplied by charger. If USB host/hub are connected, the maximal 1 00mA current is supplied to the device, the device should be able to power up to boot the image with less than 100mA.

If LPB\_BOOT fuses are blown, the Chip will check if there is low power condition via GPIO\_3 pad. If there is low power boot condition USB charger detection will be activated. If there is no USB charger, ROM will initialize USB as device and apply division factors on ARM, DDR, AXI and AHB root clocks based on LPB\_BOOT fuses value (see the table below). Polarity of low power boot condition on GPIO\_3 pad is set by BT\_LPBO\_Polarity fuse (see the figure below).

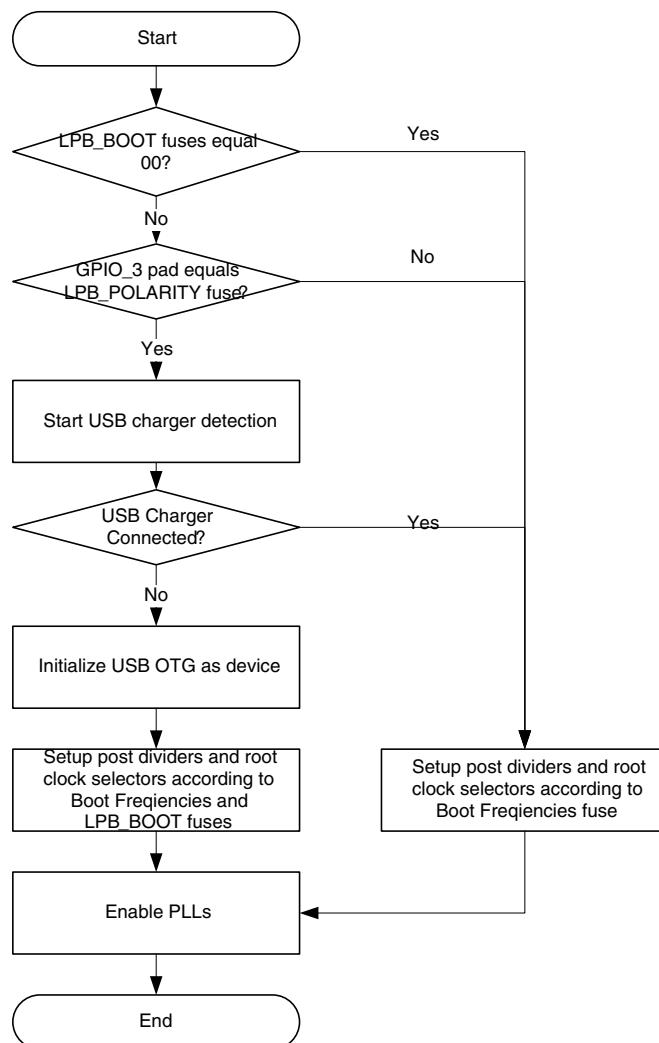
**Table 4-66. USB Low Power Boot Frequencies**

LPB_BOOT	Boot Frequencies=0	Boot Frequencies=1
00	ARM_CLK_ROOT=792MHz MMDC_CH0_AXI_CLK_ROOT=528MHz MMDC_CH1_AXI_CLK_ROOT=528MHz AXI_CLK_ROOT=264MHz AHB_CLK_ROOT=132MHz	ARM_CLK_ROOT=396MHz MMDC_CH0_AXI_CLK_ROOT=352MHz MMDC_CH1_AXI_CLK_ROOT=352MHz AXI_CLK_ROOT=176MHz AHB_CLK_ROOT=88MHz
01	ARM_CLK_ROOT=792MHz MMDC_CH0_AXI_CLK_ROOT=528MHz MMDC_CH1_AXI_CLK_ROOT=528MHz AXI_CLK_ROOT=264MHz AHB_CLK_ROOT=132MHz	ARM_CLK_ROOT=396MHz MMDC_CH0_AXI_CLK_ROOT=352MHz MMDC_CH1_AXI_CLK_ROOT=352MHz AXI_CLK_ROOT=176MHz AHB_CLK_ROOT=88MHz

*Table continues on the next page...*

**Table 4-66. USB Low Power Boot Frequencies  
(continued)**

10	ARM_CLK_ROOT=396MHz MMDC_CH0_AXI_CLK_ROOT=264MHz MMDC_CH1_AXI_CLK_ROOT=264MHz AXI_CLK_ROOT=132MHz AHB_CLK_ROOT=66MHz	ARM_CLK_ROOT=264MHz MMDC_CH0_AXI_CLK_ROOT=176MHz MMDC_CH1_AXI_CLK_ROOT=176MHz AXI_CLK_ROOT=88MHz AHB_CLK_ROOT=44MHz
11	ARM_CLK_ROOT=264MHz MMDC_CH0_AXI_CLK_ROOT=132MHz MMDC_CH1_AXI_CLK_ROOT=132MHz AXI_CLK_ROOT=66MHz AHB_CLK_ROOT=66MHz	ARM_CLK_ROOT=132MHz MMDC_CH0_AXI_CLK_ROOT=88MHz MMDC_CH1_AXI_CLK_ROOT=88MHz AXI_CLK_ROOT=44MHz AHB_CLK_ROOT=44MHz

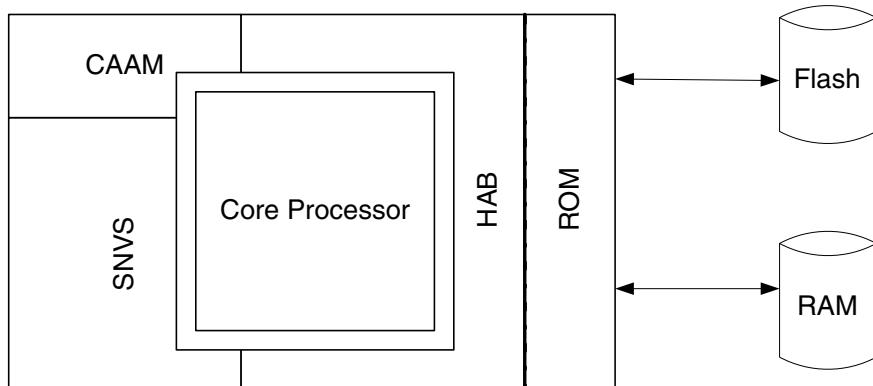


**Figure 4-24. USB Low Power Boot Flow**

## 4.11 High Assurance Boot (HAB)

The High Assurance Boot (HAB) component of the ROM protects against the potential threat of attackers modifying areas of code or data in programmable memory to make it behave in an incorrect manner. The HAB also prevents attempts to gain access to features which should not be available.

The integration of the HAB feature with the ROM code ensures that Chip does not enter an operational state if the existing hardware security blocks have detected a condition that may be a security threat or areas of memory deemed to be important have been modified. The HAB uses RSA digital signatures to enforce these policies.



**Figure 4-25. Secure Boot Components**

The figure above illustrates the components used during a secure boot using HAB. The HAB interfaces with the SNVS to ensure the system security state is as expected. The HAB also makes use of CAAM hardware block to accelerate SHA-256 message digest operations performed during signature verifications and AES-128 operations for encrypted boot operations. The HAB also includes a software implementation of SHA-256 for cases where a hardware accelerator cannot be used. The RSA key sizes supported are 1024, 2048 and 3072 bits. The RSA signature verification operations are performed by a software implementation contained in the HAB library. The main features supported by HAB are:

- X.509 Public key certificate support
- CMS signature format support

- Proprietary encrypted boot support. Note that encrypted boot depends on the CAAM HW module. When CAAM is disabled (i.e. when the EXPORT\_CONTROL fuse is blown) then encrypted boot is not available.
- Encrypted Boot Support

**NOTE**

Freescale provides a reference Code Signing Tool (CST) for key generation and code signing for use with the HAB library. The CST can be found by searching for "IMX\_CST\_TOOL" at: <http://www.freescale.com>.

**NOTE**

For further details on making use of the secure boot feature using HAB contact your local Freescale representative.

### 4.11.1 ROM Vector Table Addresses

For devices that perform a secure boot, the HAB library may be called by boot stages that execute after ROM code. The RVT table contains the pointers to the HAB API functions and is located at 0x00000094.

**NOTE**

For additional information on secure boot including the HAB API, contact your local Freescale representative.



# **Chapter 5**

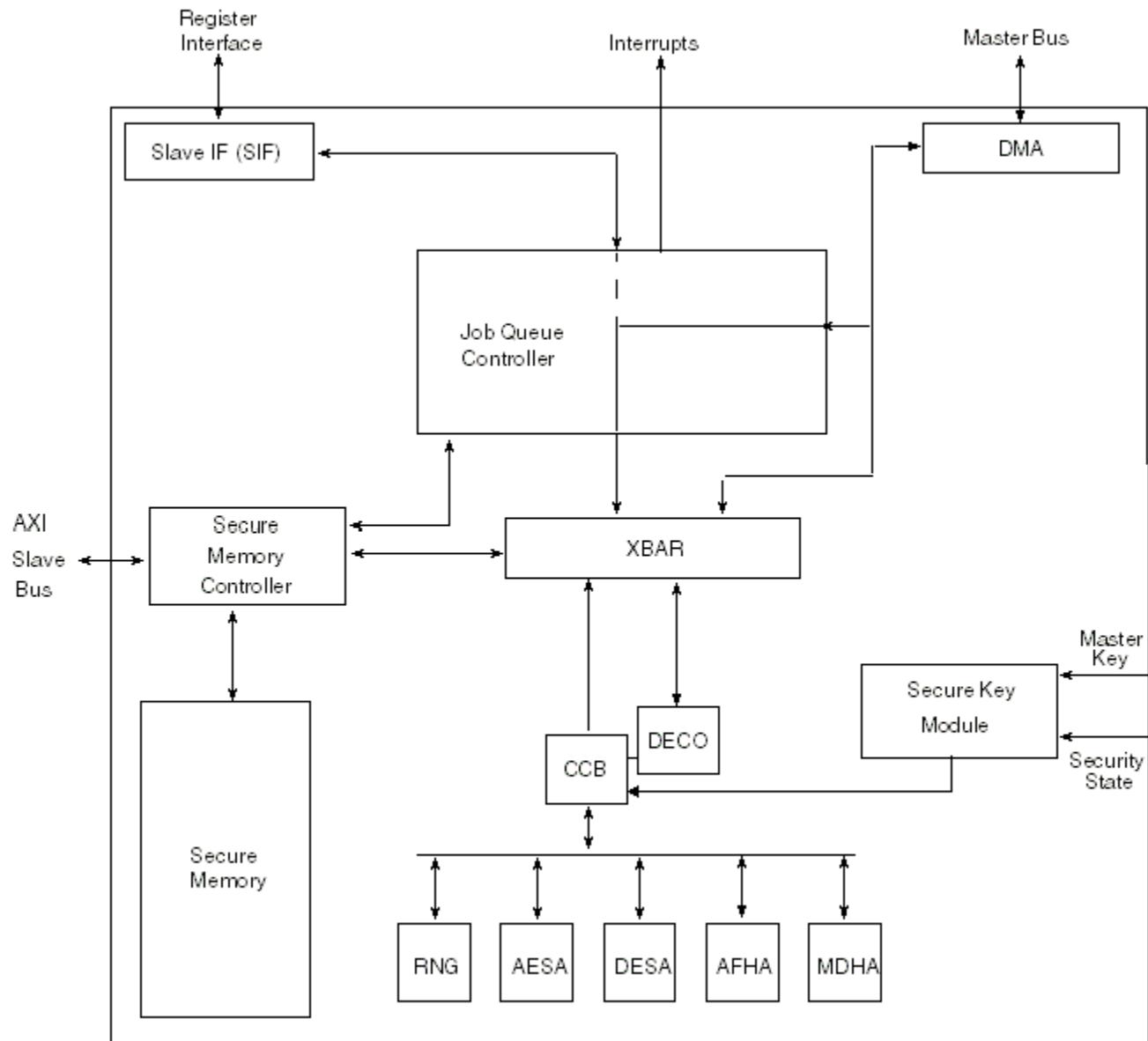
## **Cryptographic Acceleration and Assurance Module (CAAM)**

### **5.1 Overview of CAAM ( cryptographic acceleration and assurance module) functionality**

CAAM is the chip's cryptographic acceleration and assurance module, which serves as Freescale's latest cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It also implements block encryption algorithms, stream cipher algorithms, hashing algorithms, a secure memory controller, and a hardware random number generator. CAAM includes the following interfaces:

- A slave bus interface for the processor to write configuration and command information, and to read status information
- Two bus interfaces. The first is a slave interface that gives access to the secure memory and the other is a master interface that allows CAAM to read/write data from external memory

This figure shows the block diagram.

**Figure 5-1. CAAM block diagram**

## 5.2 Feature summary

CAAM includes the following features:

- Offloading of cryptographic functions via programmable job descriptor language
  - Job descriptor can contain multiple function commands.
  - Job descriptor can be chained to additional job descriptors.
  - Job descriptors are submitted via 2 separate hardware-implemented job rings.
  - Each job ring implements access controls.
- Register bus interface

- 32-bit read/write slave bus
- DMA
  - 32-bit DMA interface
  - Automatic byte and half-word ordering
  - Scatter/gather support for data
- Secure memory
  - 32-bit AXI Slave Bus Interface
  - 1 default partition, plus 7 optional partitions
    - Each partition can be owned by any processor
    - Partitions are variable-sized: zero or more pages
    - Pages can be dynamically allocated to or de-allocated from partitions
  - Access control per partition
    - Partition owner can allow/disallow access to partition by specific bus masters
    - Partition can be designated as accessible via read and/or write transactions
    - Partition can be designated as accessible only via key-read job descriptor transactions
    - Partition can be designated as accessible via job-ring-specific trusted descriptors
    - Partition can be designated as permitting or not permitting export and import of secure memory blobs
  - Automatic Zeroization of secure memory
    - Zeroization on reset, failure, and requested de-allocation of pages or partitions
    - Partitions can be excluded from automatic zeroization
- Secure key module
  - Black keys
    - Keys stored in memory in encrypted form and decrypted on-the-fly when used
    - AES\_ECB or AES\_CCM encryption using a 256-bit key
  - Export and import of cryptographic blobs
    - Data encapsulated in a cryptographic data structure for storage in nonvolatile memory
    - AES\_CCM encryption using a 256-bit key
    - Each blob encrypted using its own randomly generated blob key.
    - Blob key encrypted using a non-volatile blob key encryption key
    - Blob key encryption key derived from non-volatile master key input
    - General memory blob key encryption key derived from non-volatile master key input

- Secure memory blob key encryption key derived from partition access permission bits and non-volatile master key input
- Separate blob key encryption keys for trusted mode, secure mode, and non-secure mode
- Cryptographic authentication
  - Hashing algorithms
    - MD5
    - SHA-1
    - SHA-224
    - SHA-256
  - Message authentication codes (MAC)
    - HMAC-all hashing algorithms
    - AES-CMAC
    - AES-XCBC-MAC
  - Auto padding
  - ICV checking
- Authenticated encryption algorithms
  - AES-CCM (counter with CBC-MAC)
- Symmetric key block ciphers
  - AES (128-bit, 192-bit or 256-bit keys)
  - DES (64-bit keys, including key parity)
  - 3DES (128-bit or 192-bit keys, including key parity)
  - Cipher modes
    - ECB, CBC, CFB, OFB for all block ciphers
    - CTR for AES
- Symmetric key stream ciphers
  - ArcFour (alleged RC4 with 40 - 128 bit keys)
- Random-number generation
  - Entropy is generated via an independent free running ring oscillator
  - Oscillator is off when not generating entropy; for lower-power consumption
  - NIST-compliant, pseudo random-number generator seeded using hardware-generated entropy

## 5.3 CAAM implementation

CAAM provides platform assurance by working with SNVS, which is a companion logic block that tracks the security state of the chip. CAAM is programmed by means of descriptors that indicate which operations should be performed and point to the message and associated data. CAAM incorporates a DMA engine to fetch the descriptors, read the

message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that CAAM can read and write data scattered in memory. CAAM may be configured by means of software for dynamic changes in byte ordering.

### 5.3.1 CAAM submodules

The CAAM core contains the following submodules:

- Master bus interface
- Slave bus interface
- Register bus interface
- Job queue controller (JQC)
- Descriptor controller (DECO)
- Cryptographic control block (CCB)
- Multiple cryptographic hardware accelerators (CHAs)
- Secure memory

JQC fetches descriptors that tell CAAM which cryptographic operations to perform and on what data to operate. DECO decodes descriptors and executes the commands within them. For those descriptor commands that use CHAs, DECO communicates with the CHAs by means of the CCB.

### 5.3.2 Cryptographic engines implemented in CAAM

The cryptographic engines provided are:

- Data encryption standard (DES) accelerator (DESA)
- Advanced encryption standard (AES) accelerator (AESA)
- Message digest (for example, hashing) hardware accelerator (MDHA)
- Random-number generator (RNG)
- ARC four (alleged RC4) hardware accelerator (AFHA)

## 5.4 CAAM modes of operation

CAAM can operate in the following security modes:

- Trusted
- Secure
- Non-secure
- Fail

## CAAM modes of operation

SNVS controls these modes based on SNVS's current security state (that is, init, check, trusted, secure, non-secure, and fail). The primary difference between these modes is that they have different cryptographic keys available. Within each mode there are keys that are volatile (that is, a different key value is used for each power-on session) and keys that are non-volatile (that is, the same key value is available during each power-on session).

### 5.4.1 Security Monitor security states

The current security mode can be identified in the CAAM status register's (CSTA's) MOO field.

This figure shows an overview of the SNVS security state transitions. Note that CAAM can detect certain security alarm conditions and can signal an alarm to SNVS. Depending upon the settings and configuration of SNVS, this may cause the SNVS security state machine to transition to fail state, which would then put CAAM into fail mode.

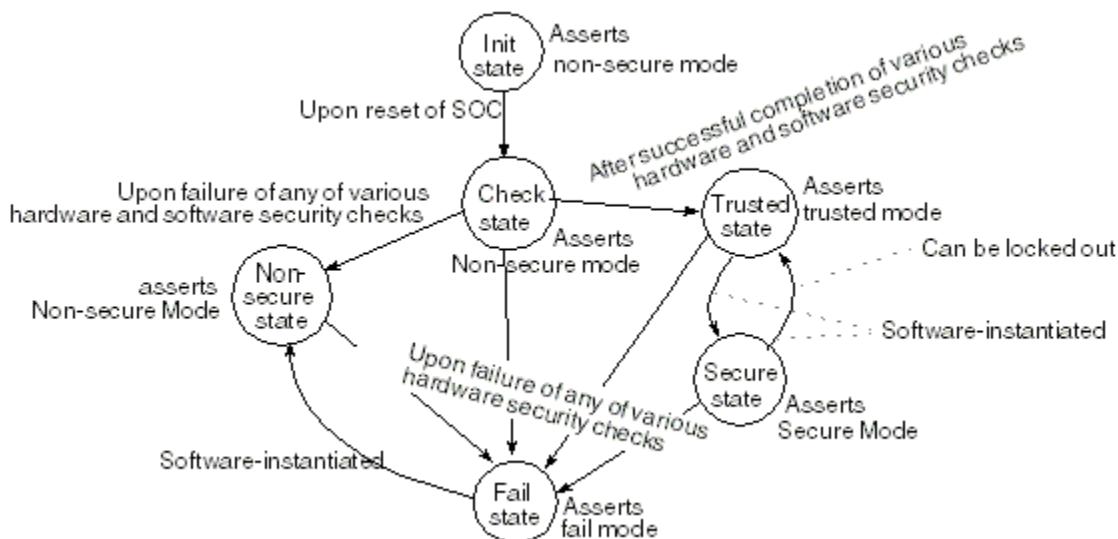


Figure 5-2. SNVS security state machine diagram

#### 5.4.1.1 The effect of security state on volatile keys

At each power-on reset CAAM generates new random values for three 256-bit volatile cryptographic keys. The secret values for these keys are generated by CAAM's RNG and then stored within secure registers in CAAM. The values are zeroized when CAAM transitions to fail mode (in other words, when the SNVS's security state machine transitions to fail state).

The available volatile keys are as follows:

- Job descriptor key encryption key (JDKEK) - used by job descriptors for encrypting black keys (encrypted keys)
- Trusted descriptor key encryption key (TDKEK) - used by trusted descriptors for encrypting black keys
- Trusted descriptor signing key (TDSK) - used to authenticate trusted descriptors (digitally signed job descriptors)

Note that the JDKEK, the TDKEK, and the TDSK are all available for use by CAAM in trusted mode, secure mode, and non-secure mode, but this does not cause any security issue. The reason that this is not a security issue is that the trusted mode and secure mode are intended to use the same values for these keys, and these key values are different when in non-secure mode (which is not allowed to obtain the trusted/secure state mode values of these keys). The reason that CAAM cannot obtain the trusted/secure state mode values of these keys when in non-secure mode is that new values for these keys are generated by CAAM's hardware RNG at each POR, and these keys are zeroized when entering fail mode. As shown in [Figure 5-2](#), the only paths from trusted state or secure state to non-secure state pass through fail state, and there is no path from non-secure state to either trusted state or secure state. Consequently, when operating in non-secure mode CAAM does not have access to previous trusted mode/secure mode values of these keys.

#### 5.4.1.2 The effect of security state on non-volatile keys

Data that must be retained when the system is powered off must be stored in non-volatile storage. Some of this data is disclosure-sensitive (such as data rights management keys) and must be protected even when the system is powered off. CAAM implements non-volatile cryptographic keys that can be used to encrypt sensitive data during one power-on cycle, and then decrypt it during a subsequent power-on cycle. These non-volatile keys (blob key encryption keys) are derived from the master key input that CAAM receives from SNVS.

When CAAM is operating in trusted mode or secure mode, CAAM derives general memory blob key encryption keys (GM BKEKs) and secure memory blob key encryption keys (SM BKEKs) from its master key input. When CAAM is operating in non-secure mode or fail mode, BKEKs are derived from the non-volatile test key (NVTK), a hardwired constant used for known-answer testing.

The CAAM modes are explained further in [Keys available in different security modes](#).

## 5.4.2 Keys available in different security modes

The primary difference between CAAM's security modes is that different cryptographic keys available are available in the different modes. See each mode's section for the description of the mode's special keys.

### 5.4.2.1 Keys available in trusted mode

While in trusted mode, CAAM can use special keys as listed in this table.

**Table 5-1. Special keys used in trusted mode**

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> <li>At POR, a new value (shared with secure mode but not shared with non-secure mode) is generated from RNG</li> <li>Zeroized when entering fail mode</li> </ul>	Used for automatic key encryption and decryption when executing Job Descriptors, Trusted Descriptors and Shared Descriptors
Trusted descriptor key encryption key		Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors.
Trusted descriptor signing key		Used for signing, verifying and re-signing Trusted Descriptors
GM BKEK trusted mode value	Non-volatile, not shared with secure mode or non-secure mode	Used for general memory blob encapsulation or decapsulation operations
SM BKEK trusted mode value	Non-volatile, not shared with secure mode or non-secure mode	Used for secure memory blob encapsulation or decapsulation operations

### 5.4.2.2 Keys available in secure mode

While in secure mode, CAAM can use special keys as listed in this table.

**Table 5-2. Special keys used in secure mode**

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> <li>At POR a new value (shared with trusted mode but not shared with non-secure mode) is generated from RNG</li> <li>Zeroized when entering fail mode</li> </ul>	Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors and shared descriptors
Trusted descriptor key encryption key		Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors
Trusted descriptor signing key		Used for signing, verifying and re-signing Trusted Descriptors
GM BKEK secure mode value	Non-volatile, not shared with trusted mode or non-secure mode)	Used for general memory blob encapsulation or decapsulation operations

*Table continues on the next page...*

**Table 5-2. Special keys used in secure mode (continued)**

Key	Characteristic(s)	Function(s)
SM BKEK secure mode value	Non-volatile, not shared with trusted mode or non-secure mode	Used for secure memory blob encapsulation or decapsulation operations

### 5.4.2.3 Keys available in non-secure mode

In non-secure mode a fixed default key with a known value (the NVTK) is used in place of the master key. This allows the cryptographic blob mechanism to be tested using known test results. The key registers are read and write accessible in non-secure mode, which allows testing using known test results. While in non-secure mode CAAM can use special keys as listed below.

**Table 5-3. Special keys used in non-secure mode**

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> <li>At POR, a new value (not shared with trusted mode or secure mode) is generated from RNG</li> <li>Zeroized when entering fail mode</li> </ul>	<ul style="list-style-type: none"> <li>Can be read and overwritten for testing</li> <li>Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors, and shared descriptors</li> </ul>
Trusted descriptor key encryption key		
Trusted descriptor signing key		
GM BKEK known test value	Non-volatile, not shared with trusted mode or non-secure mode)	Used for testing general memory blob encapsulation or decapsulation operations
SM BKEK known test value	Non-volatile, not shared with trusted mode or secure mode	Used for testing secure memory blob encapsulation or decapsulation operations

### 5.4.2.4 Keys available in fail mode

When CAAM transitions to fail mode, CAAM clears all registers that could potentially hold sensitive data. Because of this, cryptographic operations that are in progress when the transition occurs may not produce the correct result. If this is the case, the operation completes with an error indication.

## CAAM hardware functional description

Although CAAM cleans up ongoing operations after a transition to fail mode, CAAM is not intended to continue operating in this mode. The key registers are still usable as in the trusted and secure modes, but these registers contain all-zero values. Software can initiate a transition from fail mode to non-secure mode by commanding the SNVS security state machine to transition from fail state to non-secure state (unless this transition has previously been locked out via software). However, since all key registers and all CSP partitions were cleared when CAAM entered fail mode, the only useful actions that can be performed after the transition to non-secure mode would be those required to investigate the cause of the transition to Fail mode.

Note that is not possible to transition from fail mode to secure mode or trusted mode.

## 5.5 CAAM hardware functional description

As shown in [Figure 5-1](#), CAAM functionality is aligned with several major subcomponents. This table describes these subcomponents.

**Table 5-4. CAAM subcomponents**

Description	Cross-reference(s)
<b>Interfaces</b>	
Register interface	Register interface (IP bus)
Primarily used for access to control and status registers	—
Job execution interfaces <ul style="list-style-type: none"><li>• Job ring interface</li></ul>	<ul style="list-style-type: none"><li>• Job ring interface</li></ul>
<b>Job Queue Controller</b>	
Schedules tasks to the cryptographic acceleration modules	—
<b>Descriptor processor</b>	
Descriptor controller (DECO)	Descriptors and descriptor commands
Cryptographic control block (CCB)	Descriptor controller (DECO) and cryptographic control block (CCB)
<b>Cryptographic hardware accelerators (CHAs)</b>	
Alleged RC4 hardware accelerator (AFHA)	ARC-4 hardware accelerator (AFHA) CHA functionality
DES and 3DES hardware accelerator (DESA)	Data encryption standard accelerator (DES) functionality
Random Number Generator (RNG)	Random-number generator (RNG) functionality
Message Digest Hardware Accelerator (MDHA)	Message digest hardware accelerator (MDHA) functionality
AES Hardware Accelerator (AES)	AES accelerator (AES) functionality
<b>Trust Architecture modules</b>	
Secure memory	Secure memory
Secure key module	<ul style="list-style-type: none"><li>• Black keys</li><li>• Blobs</li><li>• Trusted descriptors</li></ul>

## 5.5.1 Buses

CAAM uses the following buses:

- AXI master bus (DMA interface, see [DMA interface \(AXI master bus\)](#))
- AXI slave bus (Secure Memory interface, see [Secure memory interface \(AXI slave bus\)](#))
- IP bus (register interface, see [Register interface \(IP bus\)](#))

The interfaces are CAAM's connection to the rest of the chip and to system memory; see [Figure 5-1](#).

### 5.5.1.1 DMA interface (AXI master bus)

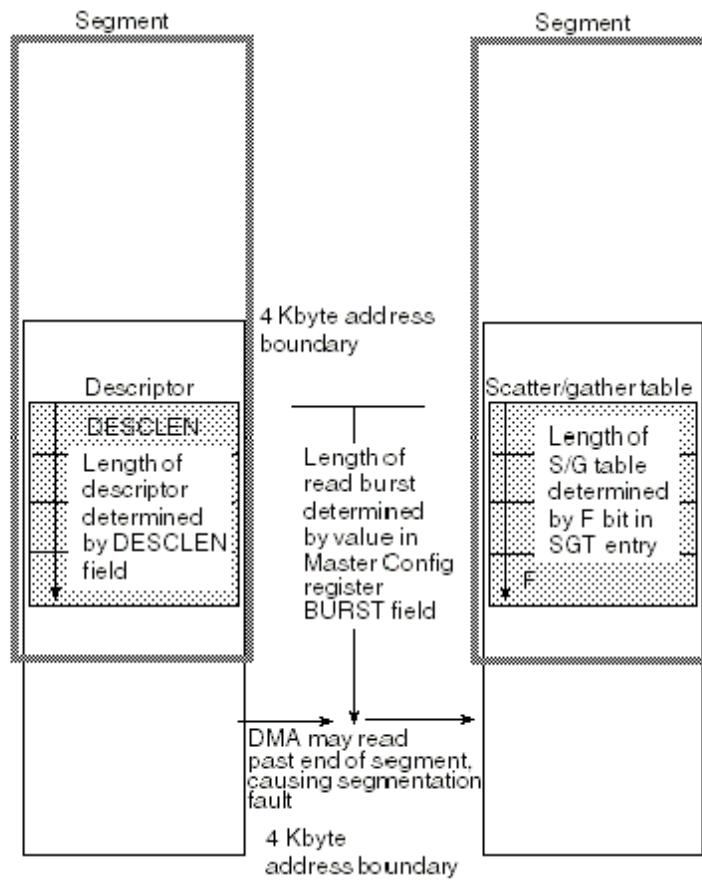
CAAM uses a DMA controller to read and write data over a 32-bit AXI bus master interface. The DMA can access system memory at byte-level granularity.

Note that to improve performance, the DMA reads and writes data in bursts, whenever possible. A normal burst is 32 bytes or 64 bytes, as indicated by the BURST field of the Master Configuration Register (MCFGR), but the DMA can be configured to use larger bursts for some accesses by setting the LARGE\_BURST bit in MCFGR (see the Master Configuration Register (MCFGR) for further details).

#### 5.5.1.1.1 DMA bursts that may read past the end of data structures

In keeping with the AXI specification, DMA accesses do not read a full burst if the read would need to cross a 4 Kbyte address boundary. CAAM also does not read a full burst from a job ring input or output ring if it would need to read past the end of the ring.

However, as illustrated in [Figure 5-3](#), CAAM may read past the end of a descriptor or scatter/gather table (SGT) when fetching them because it may not know the length of the descriptor or SGT before issuing the read transaction. Be aware of this fact if a short descriptor or SGT is stored near the end of an access control region (for example, a memory segment) that does not end at a 4 Kbyte boundary.

**Figure 5-3. DMA may read past end of descriptor or SGT**

### 5.5.1.2 Secure memory interface (AXI slave bus)

The Secure Memory within CAAM is accessible to external bus masters over a 32-bit AXI bus slave interface. The interface allows any word within Secure Memory to be accessed at byte-level granularity unless blocked by the Secure Memory access controls.

### 5.5.1.3 Register interface (IP bus)

CAAM's register interface (32-bit IP bus) is used to read and write registers within CAAM for the following purposes:

**Table 5-5. Summary of register interface uses**

Purpose	For more information, see
During chip initialization time	
To configure CAAM, including initialization of the job rings	<a href="#">Initializing job rings</a>
Change the default settings for CAAM's AXI DMA interface	Master Configuration Register (MCFGR) in the memory map

*Table continues on the next page...*

**Table 5-5. Summary of register interface uses (continued)**

Purpose	For more information, see
Configure Secure Memory	<a href="#">Initializing secure memory</a>
	During normal steady-state operations
Manage CAAM's job ring interface	<a href="#">Job ring interface</a>
Alter the Secure Memory configuration	<a href="#">Secure memory</a>
	During hardware and software debugging
Read status registers and single-step descriptor commands	<ul style="list-style-type: none"> <li>• <a href="#">Executing job descriptors in single-step mode</a></li> <li>• CAAM Status Register</li> <li>• RNG TRNG Status Register</li> <li>• RNG DRNG Status Register</li> <li>• Holding Tank Status Register</li> <li>• Secure Memory Status Register</li> <li>• Job Ring Output Status Register</li> <li>• Job Ring Interrupt Status Register</li> <li>• CCB Interrupt Status Register</li> <li>• CCB Status and Error Register</li> <li>• CCB FIFO Status</li> <li>• DECO Operation Status Register</li> </ul> <p>For all registers, see the "CAAM Register Descriptions" appendix.</p>

**NOTE**

Accesses to registers other than the Input Data FIFO must use full-word (32-bit) reads or writes. Writing individual bytes to the Input Data FIFO is permitted.

## 5.5.2 Job execution interfaces

CAAM provides cryptographic services by executing a series of jobs defined by job descriptors or trusted descriptors. Unless otherwise specified, any reference to job descriptors applies to trusted descriptors as well. Each job descriptor is a small program of CAAM commands. The commands may include conditional branches, loops, subroutine calls, or jumps to other job descriptors, as well as mathematical, cryptographic and data movement operations.

When software creates a job descriptor, it uses one of CAAM's job rings to submit the job to CAAM. This interface is described in [Job ring interface](#).

Job descriptors submitted through job rings reside in memory and are automatically fetched by CAAM's DMA when it is their turn for execution. Although the DMA can fetch job descriptors from any byte offset, for performance reasons it is recommended that descriptors always begin on cache line boundaries.

For further information, see [Job descriptors](#) and [Trusted descriptors](#).

### 5.5.2.1 Job ring interface

CAAM implements two job ring interfaces. Each job ring interface implements an input ring for job descriptors and an output ring for results. This queueing mechanism allows software to schedule multiple CAAM jobs for execution and then retrieve the results as convenient. These input rings and output rings are implemented as circular buffers (also called rings) that may be located in system memory or they may be located in CAAM Secure Memory.

The entries in the input rings are pointers to job descriptors that are located elsewhere in memory (see [Address pointers](#)). Each entry in an output ring consists of a pointer to a job descriptor followed by a job termination status/error word (see [Job termination status/error codes](#)). They may also be followed by a word containing the number of bytes written by SEQ STORE and SEQ FIFO STORE commands during the descriptor's execution (see INCL\_SEQ\_OUT field in the Job Ring Configuration Register). The job descriptor pointers in the output rings allow software to correlate result status with the particular job descriptor that CAAM executed to produce that result.

#### 5.5.2.1.1 Configuring and managing the input/output rings overview

Software configures the input rings and output rings and then manages them jointly with CAAM. The following table describes the uses of the input and output ring registers:

**Table 5-6. Input/output ring registers**

Register	Description
Input/Output Ring Base Address Register	Describes the base address of the ring buffer, which must be a multiple of four bytes
Input/Output Ring Size Register	Describes the size of the ring buffer measured in the number of entries
Input Ring Jobs Added/Output Ring Jobs Removes Register	Tells CAAM how many jobs software placed in the input ring or removed from the output ring
Input Ring Slots Available/Output Ring Slots Full Register	Tells software how many spaces are available to add jobs to the input ring or how many jobs are in the output ring ready for software processing
Input Ring Read Index	Points to the head of the queue within the input ring, that is, where CAAM finds the next job descriptor to read from the job ring
Output Ring Write Index	Points to the tail of the queue within the output ring, that is, where CAAM places the results of the next completed job descriptor for that job ring

Figure 5-4 shows an example input ring and output ring. The physical ring buffers are shown in the boxes on the right. The logical queues located within these ring buffers are shown in shaded boxes to the left. Each job ring entry consists of a pointer to a job descriptor. Each output ring entry consists of a pointer to a job descriptor followed by a 32-bit word indicating the job completion status.

In this example, jobs 10 through 15 are in the job ring waiting for CAAM to process them. The results for jobs 4 through 8 are in the output ring, waiting for software to retrieve them. CAAM has removed job 9 from the job ring, but has not yet written the results to the output ring. Old entries that have not yet been overwritten are shown in italics.

Note that job 7 completed ahead of job 6. Although this version of CAAM implements only one DECO, in versions of CAAM that implement more than one DECO, it is possible for jobs submitted through the same job ring to complete out of order. See [Order of job completion](#).

## CAAM hardware functional description

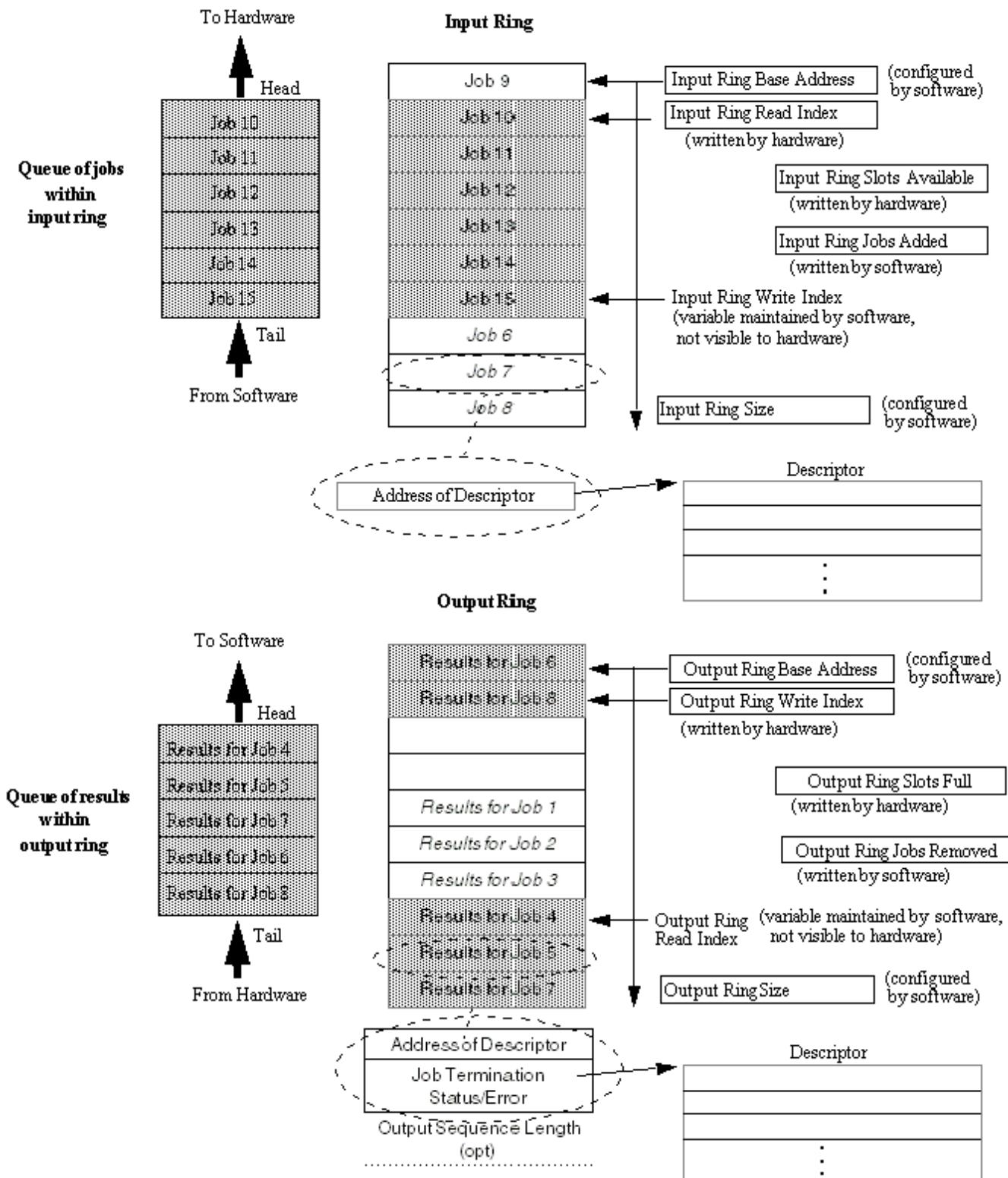


Figure 5-4. Input and output ring example

### 5.5.2.1.2 Managing the input rings

For the input ring, software is the producer, meaning that software

- Writes descriptor addresses into the job descriptor queue within the input ring.
- Writes the number of new jobs to the Input Ring Jobs Added Register

The address added to the input ring must point directly to the start of a job descriptor, not to a scatter/gather table. A job descriptor queue entry is one word. (See [Address pointers](#).) Software maintains its own write pointer for the input ring, and CAAM does not have direct access to that pointer.

For the input ring, CAAM is the consumer, meaning that it increments the Input Ring Slots Available Register upon pulling descriptor addresses out of the queue. When software writes a new value to the Input Ring Jobs Added Register, CAAM decrements the Input Ring Slots Available Register by the value that was written by software. CAAM maintains a read index that it increments as it reads jobs from the input ring.

### 5.5.2.1.3 Managing the output rings

For the output ring, the roles are reversed from the input ring. CAAM is the producer and software is the consumer.

- When CAAM adds completed jobs to the output ring within the job ring, it increments the Output Ring Slots Full Register, which tells software how many results are available for software to retrieve. An interrupt may or may not be generated, depending upon the job ring configuration at the time (for more details, see [Asserting job ring interrupts](#)).
- When software removes jobs from the output ring for processing, it writes the number of jobs removed to the Output Ring Jobs Removed Register. CAAM decrements the output ring slots full value by the new value that software wrote to the Output Ring Jobs Removed Register.

Note that each entry in the output ring consists of a job descriptor address and a job completion status word (also called a job termination status/error word). See [Job termination status/error codes](#) for the format of this status word. Therefore, the size of an entry in the output ring is the size of a pointer plus one word for status, plus an optional word containing the output sequence length. CAAM maintains an output ring write index that it increments as it places completed jobs and status into the output ring. Software can read this register to determine the current tail of the output ring.

Note that it is possible for a bus error to occur when the job queue controller is writing the completion status to the output ring. This results in an error code type 1 indication for that particular job ring. The correct response to any job ring error code 1 indication is to reset CAAM.

#### 5.5.2.1.4 Controlling access (rings)

Access to a job ring can be restricted to a single bus master by configuring CAAM's Job Ring MID Registers. Access to a job ring can be further restricted to a particular software entity because each job ring's registers are in a separate 4 Kbyte address page. An OS or a hypervisor can enforce the restrictions by means of a memory management unit.

A process with permission to access a particular job ring's registers can:

- Schedule jobs for CAAM by writing the address of a job descriptor or trusted descriptor at the tail of the queue within the input ring.
- Retrieve job completion status by reading the entry at the head of the queue within the corresponding output ring.

Each job ring can be configured so that CAAM's DMA asserts different MID values when executing jobs on behalf of that job ring. This allows slave devices or chip-level memory management units to make memory access control decisions based upon the job ring from which the job was initiated.

#### 5.5.2.1.5 Order of job completion

Job descriptors submitted through different job rings are not guaranteed to complete in the order they were submitted by software, even if they reference the same shared descriptor and use SERIAL or WAIT sharing. (See [Shared descriptors](#) for details about shared descriptors and sharing concepts). However, since CAAM has only one DECO, job descriptors submitted through the same job ring complete in the order in which they were submitted.

#### 5.5.2.1.6 Initializing job rings

Minimal configuration for CAAM operation using the job ring interface requires initializing at least one job ring by specifying the base addresses for the input ring and output ring and the size of these rings (see the Input Ring Base Address Register (IRBAR), the Output Ring Base Address Register (ORBAR), the Input Ring Size Register (IRSR), and the Output Ring Size Register (ORSR)). Most cases also require specifying the MID values associated with the job ring (see Job Ring MID Register (JRMIDR)). Job rings can also be configured for endianness by means of the register interface (see Job Ring Configuration Register (JRCFGGR)).

### 5.5.2.1.7 Asserting job ring interrupts

Each job ring interface asserts an interrupt request on a separate interrupt request line to notify the driver software that job results are available from that job ring. Note that the software context switch overhead could have a severe performance impact if interrupts were asserted for every job completion. Therefore, CAAM supports a configurable interface that allows the driver to specify how full the output ring can be before CAAM generates a job completion interrupt request. To prevent any job from waiting too long for software completion processing, the driver can also specify a time out value. This value allows CAAM to generate an interrupt if job results are available and too much time has elapsed since software last removed any completed jobs from the output ring. These values are programmed via fields described in the Job Ring Configuration Register (JRCFGR).

The job ring interrupt does not clear automatically when jobs are removed from the output ring. Software must clear the interrupts by writing to the Job Ring Interrupt Status Register. Note that one or more additional jobs can complete while software is clearing the interrupt. Depending on the interrupt coalescing settings, an additional interrupt may immediately be generated for these newly completed jobs.

### 5.5.2.2 Executing job descriptors in single-step mode

For the purposes of debugging descriptors, it is possible to execute descriptors one descriptor at a time, or even one descriptor command at a time. When executed in this mode, software can examine the content of most DECO and CCB registers after each descriptor or descriptor command completes. To execute descriptors or commands in this fashion, software must request direct use of a DECO.

To make this request, set the RQDn bit in the DECO Request Register. This RQDn bit must remain asserted during the entire time that software wants to access that DECO/CCB block directly. This indicates to the job queue controller that it should not assign any jobs to the requested DECO block. After the job queue controller sees the RQDn bit rise, it waits for the corresponding DECO block to complete any pending tasks. As soon as the DECO block is available, the job queue controller asserts the DENn bit in the DECO Request Register. When this bit is set, software can use the DECO/CCB block by submitting descriptors by means of CAAM's register interface. Note that trusted descriptors cannot be executed via this direct access mechanism.

When software is finished using the DECO/CCB block, it must clear the RQDn bit so that the DECO is available to the job queue controller for normal processing. The job queue controller then de-asserts the DENn bit, which resets the DECO and CCB modules.

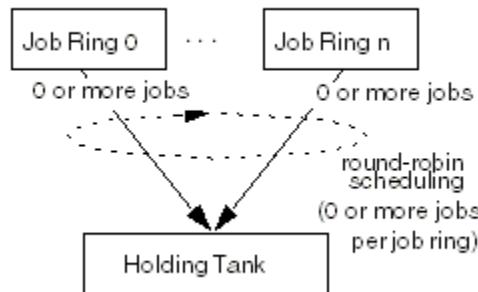
To use this feature, the DECO must be programmed in proper order so that a descriptor runs correctly. The steps are:

1. Write at least the first burst of a descriptor into the descriptor buffer. If there is a shared descriptor, offset the descriptor into the descriptor buffer by the length of the shared descriptor. If the WHL bit is not set, DECO attempts to fetch the rest of the descriptor from memory regardless of whether portions of the descriptor beyond the first burst were already written to the descriptor buffer.
2. Write the address of the descriptor into the DECO Descriptor Address Register so DECO knows where to find the descriptor. This is only required if the WHL bit is not set or if the descriptor attempts to do a STORE of type 41h, which is a write back to the descriptor in memory.
3. Write the Job Queue Control Register. If fewer than 4 words are in the first burst, the FOUR bit must be 0. If the entire descriptor has already been loaded, set the WHL bit. SHR\_FROM is not used in this format and will not be checked. When accessing a DECO directly from software, the SEQ MID and non-SEQ MID fields from the JR MID Register and Queue Interface MID Configuration Register are ignored. Instead, the DECO MID Register for the selected DECO provides the MID values that are used when the descriptor runs. Set BYTE\_SWAP to indicate whether byte swapping should be done. For further details on configuring endianness, see the following registers:
  - Job Queue Control Register
  - Job Ring Configuration Register
4. Wait until the DECO is done.
5. Read registers of interest.
6. Done or start over.

### **5.5.3 Job scheduling**

The job queue controller is the scheduler within CAAM. It pulls jobs to be processed in round-robin fashion from the job rings. Each time that the job ring's turn comes up in rotation and there is a job available in that job ring's input queue, a job is selected from that job ring. But because CAAM buffers input ring entries for efficiency, up to four jobs may be scheduled from one job ring before a job is scheduled from the next job ring. Eventually all job rings will be serviced.

The following figure illustrates the algorithm for selecting a job for an available holding tank.



**Figure 5-5. Selecting job for available holding tank**

The job queue controller prefetches some or all of the selected job descriptor and possibly the shared descriptor (if any, and it is not already in a DECO) and places them in a buffer referred to as a holding tank. After a job has been put in a holding tank, it is then eligible for dispatching to a DECO.

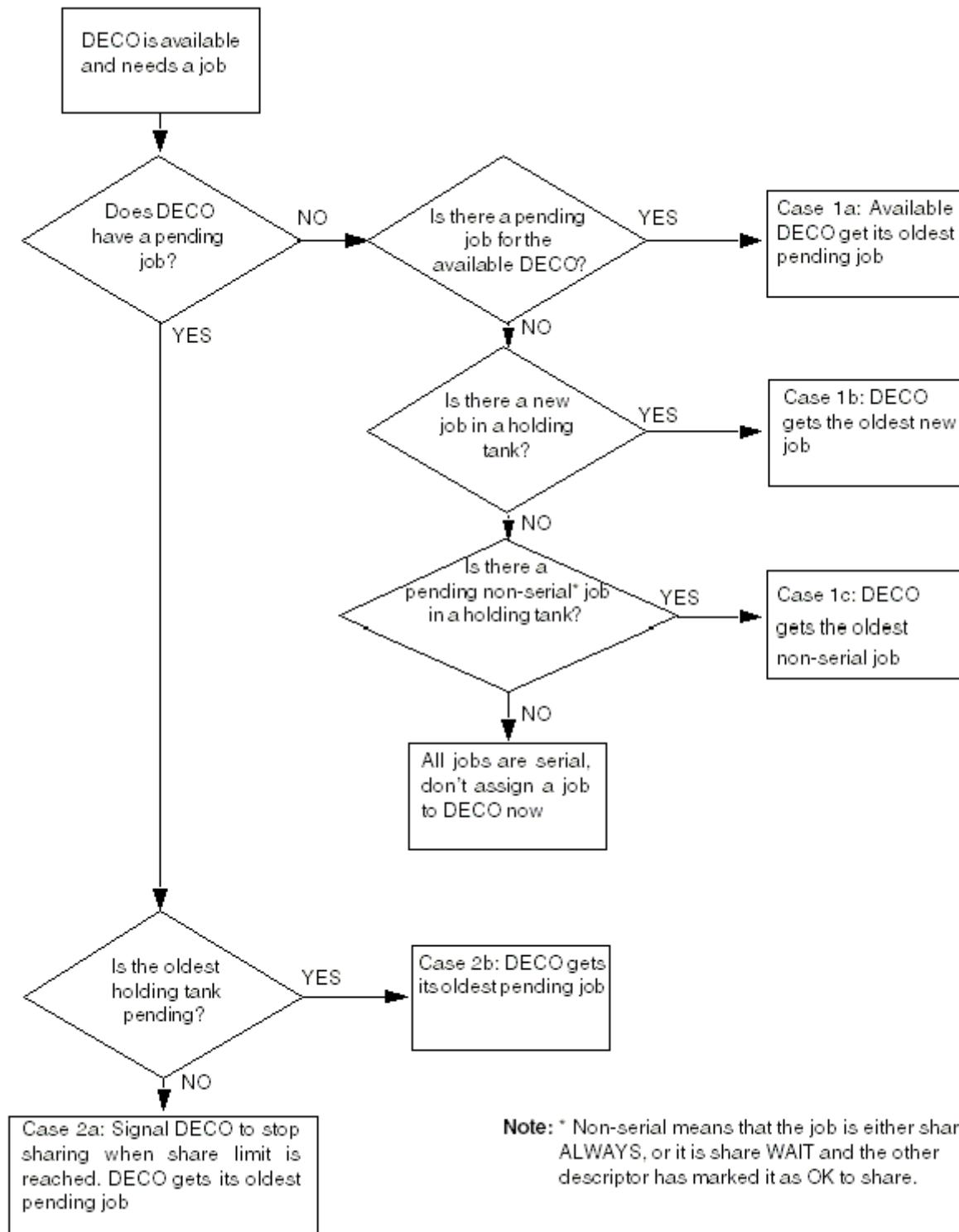


Figure 5-6. Job queue controller's job scheduling algorithm

## 5.5.4 Job termination status/error codes

CAAM reports the termination status of all jobs, allowing software to determine whether the descriptor completed normally or with some type of error. The reporting mechanism involves a job termination status/error word that is written to memory. An all-zero status indicates that the job completed without errors. If an error was encountered, bits 0-3 indicate the component within CAAM that reported the error, and the remaining bits identify the type of error (see table below). For jobs submitted through the job ring interface, this status/error word is written to the output ring in the word following the job descriptor pointer. The format of the status/error word matches the Job Ring Output Status Register.

Bits 31-28 of the status word indicate the source (within CAAM) of the job termination status/error information. The meaning of the remainder of the status word depends upon the source. The following table provides a summary of this information.

**Table 5-7. Job termination status/error codes**

Status word	Summary								
	31-28	27-24		23-20	19-16	15-12	11-8	7-4	3-0
No error	0h	0h		0h	0h	0h	0h	0h	0h
CCB	31-28	27	26-24	23-20	19-16	15-8		7-4	3-0
	2h	JMP 1	0	0	0	0h	DESC INDEX Index into the failing Descriptor	See <a href="#">Table 5-8</a>	
Jump Halt User Status	31-28	27	26-24	23-20	19-16	15-8		7-0	
	3h	JMP	0	0	0	0h	DESC INDEX Index into the failing Descriptor	OFFSET User-Defined value in the field normally used for offset	

*Table continues on the next page...*

**Table 5-7. Job termination status/error codes (continued)**

Status word	Summary							
	31-28	27	26-24	23-20	19-16	15-8		7-0
DECO	4h	JMP	0	0	0h	0h	DESC INDEX Index into the failing Descriptor	
Job Ring Status	31-28	27	26-24	23-20	19-16	15-12	11-8	7-0
	6h	0h	0h	0h	0h	0h	NADDR# descriptor addresses requested	DESC ERROR 00h No error 1Eh Error reading the Descriptor address 1Fh Error reading the Descriptor
Jump Halt Condition Codes	31-28	27	26-24	23-20	19-16	15-8		7-0
	7h	JMP	0	0	0h	0h	DESC INDEX Index into the failing descriptor	COND Condition codes field from JUMP Command

1. The descriptor made a jump to another descriptor. When this bit is set, the DESC INDEX field will contain the index into the descriptor that was jumped to and not to the original descriptor.

**Table 5-8. Bits 7-0 when status word is CCB**

7-4	3-0
CHAID 1h AES 2h DES 3h ARC4 4h MDHA 5h RNG	ERRID 0h No error 1h Mode 2h Data size 3h Key size 3h (RNG only) Instantiate 4h (RNG only) Not instantiated 5h (RNG only) Test instantiate 6h (RNG only) Prediction resistance 6h Data out-of-seq 7h (RNG only) Prediction resist & Test request 8h (RNG only) Unstantiate 9h DES key parity 9h (RNG only) Secure Key Generation Ah ICV check failed Bh Hardware error Ch CCM AAD size Dh C1 CHA not reset Eh Invalid CHA combination Fh Invalid CHA

**Table 5-9. Bits 7-0 when status word is DECO**

8-0 Part 1	8-0 Part 2
DESC ERROR	DESC ERROR
00h No error	20h DECO reset via DRR
01h SGT Length Error	21h Nonce error
03h Job Ring Control Error	23h Read Input Frame error
04h Invalid Descriptor Command	24h JDKEK, TDKEK or TDSK not loaded error
06h Invalid KEY Command	80h DNR (do not run) error
07h Invalid LOAD Command	81h undefined protocol command
08h Invalid STORE Command	C1h Undefined Blob mode
09h Invalid OPERATION Command	C2h Secure Memory Blob mode error
0Ah Invalid FIFO LOAD Command	C4h Black Blob key or input size error
0Bh Invalid FIFO STORE Command	C5h Invalid key destination in blob command
0Ch Invalid MOVE/MOVE_LEN Command	C8h Trusted/Secure mode error in blob command
0Dh Invalid JUMP Command	
0Eh Invalid MATH Command	
0Fh Invalid SIGNATURE Command	
10h Invalid Sequence Command	
11h Skip data type invalid	
12h Shared Descriptor Header Error	
13h Header Error	
14h Burster Error	
16h DMA Error	
1Ah Job failed due to JR reset	
1Bh Job failed due to Fail Mode	
1Ch DECO Watchdog timer timeout error	
1Fh MID mismatch error	

### 5.5.5 Job execution hardware

The following modules in CAAM execute cryptographic acceleration jobs:

- Descriptor controller/cryptographic control block
- Cryptographic hardware accelerators

## 5.5.5.1 Descriptor controller (DECO) and cryptographic control block (CCB)

The descriptor controller (DECO) is responsible for executing job descriptors. The DECO fetches the remainder of the job descriptor not already obtained by the job queue controller. If the job descriptor or trusted descriptor references a shared descriptor, DECO fetches the shared descriptor and begins processing. The DECO has a dedicated CCB (cryptographic control block) that it uses to access any cryptographic hardware accelerators (CHAs) needed to perform cryptographic functions.

When executing a descriptor, DECO activates the DMA controller to read the required inputs, and uses the CCB to dispatch the job to the appropriate CHAs. As data is produced by the CHAs, DECO activates the DMA to write the results and job completion status information out to the locations specified in the descriptor. When a descriptor finishes, either successfully or with errors, DECO informs the job source (job ring interface), which then takes appropriate action.

The CCB contains all the hardware necessary to control the various CHAs included in CAAM. The CCB has access to every type of CHA so that the DECO/CCB pair can perform all functions that can be performed by CAAM.

The hardware inside the CCB includes the input FIFO, output FIFO, information FIFO (NFIFO), mode registers, context registers, key registers, descriptor buffer, alignment blocks and interconnects. DECO/CCB uses all of this hardware to process descriptors.

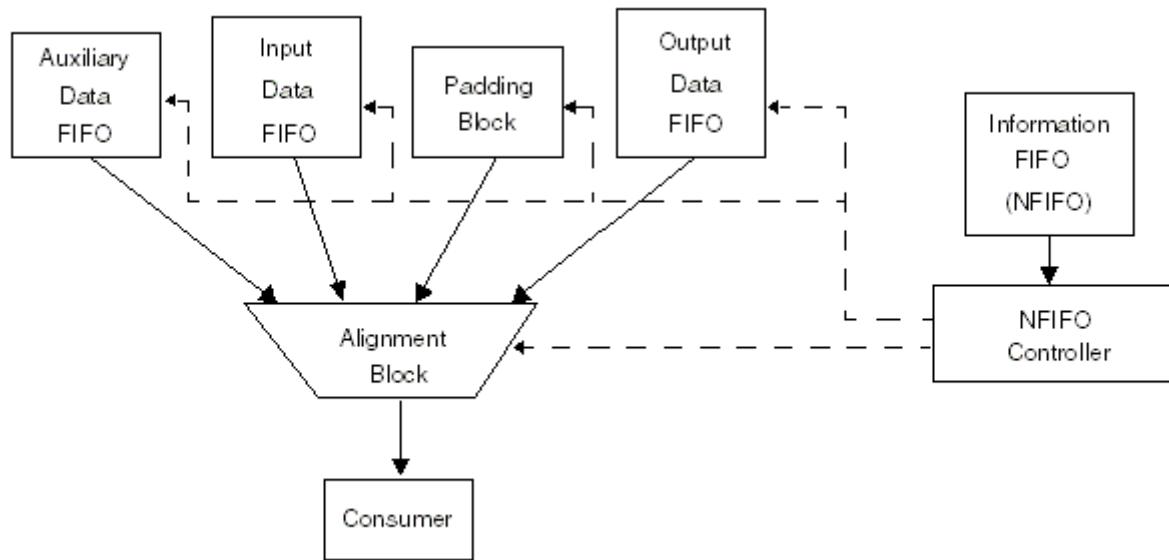
### 5.5.5.1.1 Alignment blocks

CAAM's internal data pathways and cryptographic engines generally operate on 64-bit data, but the information that CAAM obtains from memory need not be aligned to 64-bit boundaries. To concatenate and left-align information passed to certain destinations within CAAM, CAAM architecture includes three alignment blocks:

- Class 1 alignment block
- Class 2 alignment block
- DECO alignment block

Note that even if the data is aligned in memory, the alignment blocks may still need to align some portions of the data because a subset of the data may be passed to more than one destination, and the subset may need to be aligned separately for each destination.

The following figure illustrates the interconnections of one of the alignment blocks. Inputs to all three alignment blocks are identical. The blocks differ only in the consumer (Class 1 CHA, Class 2 CHA, or DECO). The entry pulled from the NFIFO tells the NFIFO controller the data source that will be used with the alignment block, and whether the alignment block will be flushed when the data transfer is complete.



**Figure 5-7. Alignment block interconnections**

All data entering Class 1 CHAs first passes through the Class 1 alignment block, which ensures that the data presented to the Class 1 CHA is properly concatenated and left-aligned. Note that the Class 1 alignment block can also serve as the source for a MOVE command (see [Table 5-37](#), SRC value 9h and AUXLS = 1) with any legal destination.

Similarly, all data entering Class 2 CHAs first passes through the Class 2 alignment block, which ensures that the data presented to the Class 2 CHA is properly concatenated and left-aligned. The Class 2 alignment block can also serve as a source for the MOVE command (see [Table 5-37](#), SRC value 9h and AUXLS = 0) with any legal destination.

The third alignment block is the DECO alignment block, which can be used as a data source for a MATH command (see [Table 5-60](#), SRC0 and SRC1 fields) and as a data source for a MOVE command (see [Table 5-37](#), SRC field).

Note that the only way to put data into an alignment block is with an info FIFO entry. Therefore, when using an alignment block as the data source for a MOVE command, the data source for the alignment block must have been specified with an info FIFO entry. The only way to take data out of the input FIFO is through one of the alignment blocks.

### 5.5.5.2 Cryptographic hardware accelerators (CHAs) (overview)

CAAM contains multiple cryptographic hardware accelerators (CHAs), each of which accelerates an encryption (Class 1) algorithm or message integrity (Class 2) algorithm.

- DESA (DES accelerator), see [Data encryption standard accelerator \(DES\) functionality](#).
- AESA (AES accelerator), see [AES accelerator \(AES\) functionality](#).
- MDHA (message digest hardware accelerator), see [Message digest hardware accelerator \(MDHA\) functionality](#).
- RNG (random number generator), see [Random-number generator \(RNG\) functionality](#).
- AFHA (ARC four hardware accelerator), see [ARC-4 hardware accelerator \(AFHA\) CHA functionality](#).

## 5.6 Descriptors and descriptor commands

Software's primary interaction with CAAM is through the creation of descriptors. To make CAAM flexible, the descriptor is a program that controls CAAM's operation. It is therefore up to the user to provide meaningful descriptors for execution. Descriptors are submitted to CAAM in order to process a job, where a job can specify a variety of functions supported by CAAM, from initialization of a security parameter, to generation of a random number, to encryption or signing of data.

Descriptors consist of commands that are executed in sequence, although conditional and unconditional jumps are available to alter the sequence. Descriptors' size is limited to 64 32-bit words. It is possible to jump from one descriptor to another so that, in effect, much larger descriptors can be created. Only the first of these descriptors has to be submitted by means of the job ring; the rest are automatically fetched and executed by CAAM.

Job descriptors, trusted descriptors and shared descriptors can be modified and written back to memory. This is done when the encryption or decryption of a data block is dependent on the operation of the prior data block. Such dependencies exist for information such as sequence numbers, counter values, and intermediate cryptographic state. Write backs are performed using descriptor commands. Hardware does not make independent decisions regarding the fields that should be written back.

### 5.6.1 Job descriptors

A job descriptor (JD) is a control structure that causes CAAM's cryptographic acceleration modules to perform a single job. Given a pointer to a job descriptor, CAAM fetches and caches the job descriptor unless a job in a holding tank uses the same job descriptor. Upon job completion, it discards the cached copy. Software generated job descriptors contain the lengths and pointers (see [Address pointers](#)) of the data to be operated on, and either directly embed security keys and context or explicitly point to

these keys and context. Explicit pointing to the keys and context can also be accomplished by pointing to a shared descriptor (SD). A job descriptor can include a shared descriptor by reference, but a shared descriptor cannot include a job descriptor.

In contrast to trusted descriptors:

- A job descriptor does not need to be created in a specially-privileged job ring.
- Job descriptors are not integrity checked.

Job descriptors use the descriptor commands defined in [Using descriptor commands](#). A job descriptor always begins with a HEADER command.

A job descriptor without a shared descriptor typically includes:

- Several commands that specify the inputs (such as keys, IV, or data) to a cryptographic operation
- Where to place the output of the operation
- An OPERATION command that specifies the operation itself

The job descriptor may contain MATH commands that perform various calculations and conditional JUMP commands that branch based upon the results of those calculations.

If the job descriptor references a shared descriptor, the memory address pointer to the shared descriptor immediately follows the job descriptor HEADER. In this case the OPERATION command and certain inputs (such as the key) are normally specified in the shared descriptor. The job descriptor typically specifies the location of the memory buffers for the input and output data. In this case the HEADER command has the REO (Reverse Execution Order) bit set so that the job descriptor commands execute first (to specify the input and output data buffers), followed by the shared descriptor commands (to specify the operations to be performed on these data buffers).

## 5.6.2 Trusted descriptors

A trusted descriptor is a job descriptor (possibly including a shared descriptor) that is integrity-checked at run time and is executed only if the check passes. This provides a mechanism to ensure that particularly sensitive operations are performed only by descriptors that were created by trusted software. Trusted descriptors have the following privileges not available to ordinary job descriptors:

- Access to trusted descriptor-only black keys (See [Blobs](#))
- Access to trusted descriptor-only blobs (See [Blobs](#))
- Access to restricted memory and device addresses
- Access to trusted descriptor-only Secure Memory partitions

Trusted descriptors allow trusted software to extend these privileges to untrusted software in a carefully controlled fashion. The trusted software can generate trusted descriptors that access specific privileged data objects in specific ways on behalf of specific requestors and deny access to other data objects, access modes, or requesters.

### **5.6.3 Shared descriptors**

Because descriptors can hold a lot of information required to process a job for a particular flow, they can be large, particularly if efficiency is maximized by placing the keys and other information within the descriptor rather than referencing them with pointers. To save overhead, CAAM supports a shared descriptor mechanism. A shared descriptor is fetched once and held internally while it is used by several different related jobs. The keys and context information can also be shared among multiple descriptors. This saves bandwidth and latency, particularly when black keys are in use.

A shared descriptor (SD) is constructed with the expectation that it will be used for multiple jobs. The general usage model is to have a shared descriptor for each security session (for example, unidirectional IPsec tunnel). Every time a job related to that security session is required, CAAM obtains job-specific information about the data (length, pointer) from the job descriptor and obtains its session context from the shared descriptor. Shared descriptors can store session state and can include commands to update this session state as needed. Shared descriptors are well suited for complex operations, as the software overhead of creating the shared descriptor is amortized over many individual jobs.

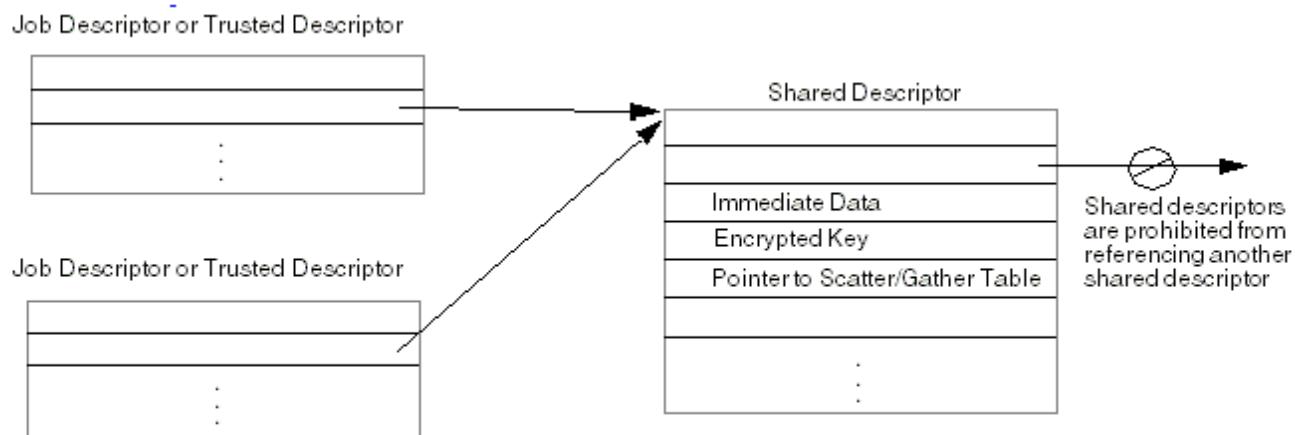
Job descriptors indicate the presence of an associated shared descriptor by setting the SHR bit in the job descriptor HEADER command. Software creates shared descriptors using the same command set as all other types of descriptors.

The following restrictions are specific to shared descriptors:

- A shared descriptor cannot have its own shared descriptor.
- A shared descriptor can be, at most, sixty-two 32-bit words. This limit is imposed because the job descriptor and the shared descriptor must both fit into the 64-word descriptor buffer (see [Figure 5-9](#)), and the minimum job descriptor consists of a one-word job descriptor HEADER and a pointer to the shared descriptor (See [Address pointers](#)). Note that larger jobs can be created by JUMPing to another job descriptor.
- Some bits in the shared descriptor and the job descriptor HEADER commands differ.
- The creation of a trusted descriptor involves signing the entire job descriptor, including a referenced shared descriptor, if any. As a result, shared descriptors are signed as part of the job descriptor when creating trusted descriptors. Therefore the

final signature is never part of a shared descriptor. Note that the REO bit cannot be set in a trusted descriptor.

The following figure illustrates two descriptors that reference the same shared descriptor.



**Figure 5-8. Two descriptors referencing the same shared descriptor**

### 5.6.3.1 Executing shared descriptors in proper order

CAAM provides mechanisms that can be used to ensure that jobs referencing the same shared descriptor execute in proper order. A shared descriptor may need to modify keys embedded within the descriptor, or particular fields of a protocol data block within the descriptor before a subsequent job uses the shared descriptor. Use the STORE command to update the shared descriptor.

A job using a shared descriptor with serial sharing does not execute until all required modifications to the shared descriptor that were initiated by prior jobs have been completed. Alternatively, the job that is asking to share a shared descriptor can use the shared wait setting to ensure that any modifications have completed before it executes that shared descriptor.

When a shared descriptor uses sequences, the sequence definitions should be in the job descriptor because the definitions can change from job descriptor to job descriptor. In such cases, set the REO bit in the shared descriptor header. Note that setting the REO bit in the shared descriptor header tells CAAM to execute the job descriptor before the shared descriptor.

### 5.6.3.2 Specifying different types of shared descriptor sharing

If two jobs are to be processed for the same data flow, they can share flow-specific data by referencing the same shared descriptor, which would be written to either reference or embed the flow-specific data. Sharing is sequential, meaning a single DECO uses the same shared descriptor to process several jobs in a row without refetching the shared descriptor. Sequential sharing is also referred to as self-sharing.

CAAM distinguishes shared descriptors from each other by the address and MID used to fetch the shared descriptor.

To share shared descriptors, the SHARE bits in the job descriptor header, and sometimes the shared descriptor header itself, must be set. This lets CAAM know under which circumstances the shared descriptors can be shared.

The following table shows the sharing possibilities supported by the HEADER command. The full details of the Shared Descriptor HEADER command can be found in [HEADER command](#).

**Table 5-10. Interpretation of the SHARE fields**

SHARE Name	Job Descriptor SHARE (binary)	Shared Descriptor SHARE (binary)	Description
NEVER	000	00	Never share the shared descriptor. Descriptors can execute in parallel, so no dependencies are allowed between them. Fetching the shared descriptor is repeated.
WAIT	001	00	Share the shared descriptor once set up has completed and processing has begun. Sharing can begin after a LOAD Command has set the OK to Share bit. Class 1 and Class 2 Key Registers are shared if valid.
SERIAL	010	00	Share once the descriptor has completed. All state is shared.
ALWAYS	011	00	Always share the shared descriptor, but keys are not shared. No dependencies can exist between the descriptors.
DEFER	100	00 NEVER 01 WAIT 10 SERIAL 11 ALWAYS	Use the value of the SHARE bits in the shared descriptor to determine the type of sharing.
All other combinations are reserved			

#### 5.6.3.2.1 Error sharing

Shared descriptors can be marked as:

- NEVER

- WAIT
- ALWAYS

When the job descriptor or shared descriptor is marked NEVER, they are not shared . In the NEVER case, the DECO fetches independent copies of the shared descriptor when executing successive jobs that reference the same shared descriptor. However, the shared descriptor being marked as NEVER indicates that it is stateless (contains no information requiring update upon completion of a job).

In a version of CAAM with a single DECO, the DECO is considered the supplier of the shared descriptor for the first job that uses the freshly-fetched shared descriptor, but is considered the consumer for subsequent jobs that use the same shared descriptor without refetching it.

In the ALWAYS case, errors do not propagate from supplier to consumers.

In the WAIT case, an error in the supplier propagates to the consumer, causing both jobs to terminate with errors. The DECO Control Register can be written with value 0300h (i.e. OFFSET = 03h and LENGTH = 00h) to block error propagation (see value 06h, class 11 in [Table 5-24](#)). Using this value sets "OK to share" and tells the supplier not to propagate its error to the consumer that is copying the shared descriptor and keys.

### 5.6.3.3 Changing shared descriptors

The best shared descriptors are independent, meaning that they do not need to be modified for each job descriptor with which they are used. Shared descriptors are more easily used the more generic they are. However, shared descriptors may have to be changed on occasion; for example, when there is a key change. Replacement job descriptors (see [Using replacement job descriptors](#)) can be used for such changes to avoid requiring software to make the change.

### 5.6.4 Using in-line descriptors

In the typical use case, the shared descriptor contains the main processing sequence. However, by setting the INL bit in a SEQ IN PTR command and providing appropriate address and length information, CAAM is directed to an in-line descriptor, which is a job descriptor that software prepends to the data defined by an input sequence (see [SEQ vs non-SEQ commands](#)). In-line descriptors are the preferred method for processing one-off jobs without using the job ring interface to submit a new job descriptor. See [SEQ IN PTR command](#) for more information about the SEQ IN PTR command.

Note that shared descriptors can point to in-line descriptors, but in-line descriptors cannot point to shared descriptors. This means that the in-line descriptor is loaded at the start of the descriptor buffer, overwriting as much shared descriptor and job descriptor as needed (if any). Note that an in-line descriptor may be scattered by means of an SGT, should that be useful.

## **5.6.5 Using replacement job descriptors**

A replacement job descriptor is an in-line descriptor that:

- Replaces the job descriptor that invoked the replacement descriptor
- Does not replace the existing shared descriptor

To invoke the replacement job descriptor, execute a SEQ IN PTR with RJD = 1. This immediately executes the replacement job descriptor. Note that the replacement job descriptor must be at the start of the input sequence data at the time that this SEQ IN PTR command is executed.

The replacement job descriptor can modify the shared descriptor before allowing it to execute. This allows operations such as changing the keys and resetting the sequence number within a shared descriptor, such as for an IPSEC PROTOCOL OPERATION). However, because the shared descriptor has already been loaded, the length and address of the shared descriptor must not be modified. Note that when there is no shared descriptor, there is no difference between an in-line descriptor and a replacement job descriptor.

When using the replacement job descriptor capability, the current job descriptor can be replaced with any job descriptor. Other data, including an input frame, can follow the replacement job descriptor in the input sequence data. For example, an IPsec flow can modify the keys or sequence number and then immediately process another packet.

If there is a JUMP HALT command in the replacement job descriptor the job terminates without executing the shared descriptor. Otherwise, if the job descriptor has the REO bit set, once the replacement job descriptor has finished, execution continues with the shared descriptor so that data can be processed. If the shared descriptor will process data during this job, before beginning that processing make sure that all the updates made to the shared descriptor have completed both internally and externally (i.e. the update to the descriptor buffer has completed and the update to the shared descriptor in memory has completed). This is discussed in the following two paragraphs.

The replacement job descriptor can insert new values in the shared descriptor with either the MOVE command or the LOAD command. The MOVE command's default behavior is to schedule the MOVE operation as soon as possible and then allow the next command

to execute. As a result, the MOVE happens in parallel with subsequent commands. Be aware that the MOVE command can take multiple cycles to complete, and it is possible that shared descriptor commands will be executed before the MOVE completes. This would result in the intended updates not being used. If there is a chance that this may occur, use the WIC bit to ensure correct operation. See [MOVE](#) and [MOVE\\_LEN commands](#) for additional details about the MOVE command.

If using the LOAD command to modify the shared descriptor, the replacement job descriptor should use the JUMP command, waiting for the NIP (No Input Pending) bit to evaluate true before proceeding. Note that the replacement job descriptor can also be used to transfer data to other destinations, such as memory, context registers, or Math registers. It is the replacement job descriptor's responsibility to ensure that any and all of these transfers have completed before the shared descriptor uses the new data.

## 5.6.6 Scatter/gather tables (SGTs)

When submitting jobs to CAAM, software can create job descriptors with address/length entries that point directly to data or indirectly to data by means of the scatter/gather tables. An SGT consists of one or more SGT entries, each of which occupies four 32-bit words, as seen in [Table 5-12](#).

Note that an SGT entry with Length = 0 is legal. When this is the setting, no data will be read from or written to the buffer pointed to by the Address Pointer.

An entry can point to another SGT that contains additional entries by setting the E (Extension) bit in the entry. When the E bit is set, CAAM fetches SGT entries from the new SGT and ignores any remaining entries in the old SGT.

### NOTE

An SGT with the E bit set in the first entry is considered malformed

The final entry in the SGT is marked by setting the F (Final) bit in an entry.

The following table lists the SGT field definitions.

**Table 5-11. Scatter/gather table field descriptions**

Field	Description
Offset	Offset (measured in bytes) into memory where significant data is to be found. The use of an offset permits reuse of a memory buffer without recalculating the address.
Length	Length of significant data in buffer, measured in bytes
F	Final Bit. If set, this entry in the scatter/gather table is the last.

*Table continues on the next page...*

**Table 5-11. Scatter/gather table field descriptions (continued)**

Field	Description
E	Extension bit. If set to a 1, the address pointer points to a scatter/gather table entry instead of a memory buffer. In this case the Length and Offset fields are ignored, and this entry is regarded as the last entry of the current scatter/gather table even if the F bit is 0. The next table entry is taken from the scatter/gather table pointed to by the address pointer. <b>NOTE:</b>
Address Pointer	Pointer to memory buffer or discontinuous scatter/gather table entry, depending on the E bit. See <a href="#">Address pointers</a>
Reserved	Field not currently defined. Leave these bits 0 to ensure forward compatibility.

**Table 5-12. Scatter/gather table entry format**

word 0	Reserved (must be 0)		
word 1	Address Pointer		
word 2	E    F    Length [ 29:0]		
word 3	Reserved (must be 0)		Offset(12:0)

## 5.6.7 Using descriptor commands

Descriptors contain one or more commands that tell CAAM what operations to perform, as well as what data on which to operate. Commands can also be used to enforce data type separation. For example, specifying that input data be treated as a cryptographic key forces CAAM to treat it exclusively as a key and prevents the key from being written back out into memory in unencrypted form.

CAAM permits a great deal of flexibility in composing descriptors, but it is highly recommended that descriptors be modeled after the examples provided in Freescale's reference software. Some sequences of commands or combinations of command options may produce unexpected results.

### 5.6.7.1 Command execution order

#### NOTE

In the following discussion, the term "Job Descriptor" should be taken to include both job descriptors and trusted descriptors.

Before a job descriptor begins execution, the portion of the job descriptor contained in the holding tank is loaded into the descriptor buffer. This includes the job descriptor's HEADER command (see [HEADER command](#)), which is the first command executed. Once the remainder of the job descriptor has finished loading, the next command to execute depends upon three fields in the HEADER: SHR, REO, and START INDEX.

The following figure shows the layouts for job descriptors depending on whether SHR = 0 or 1.

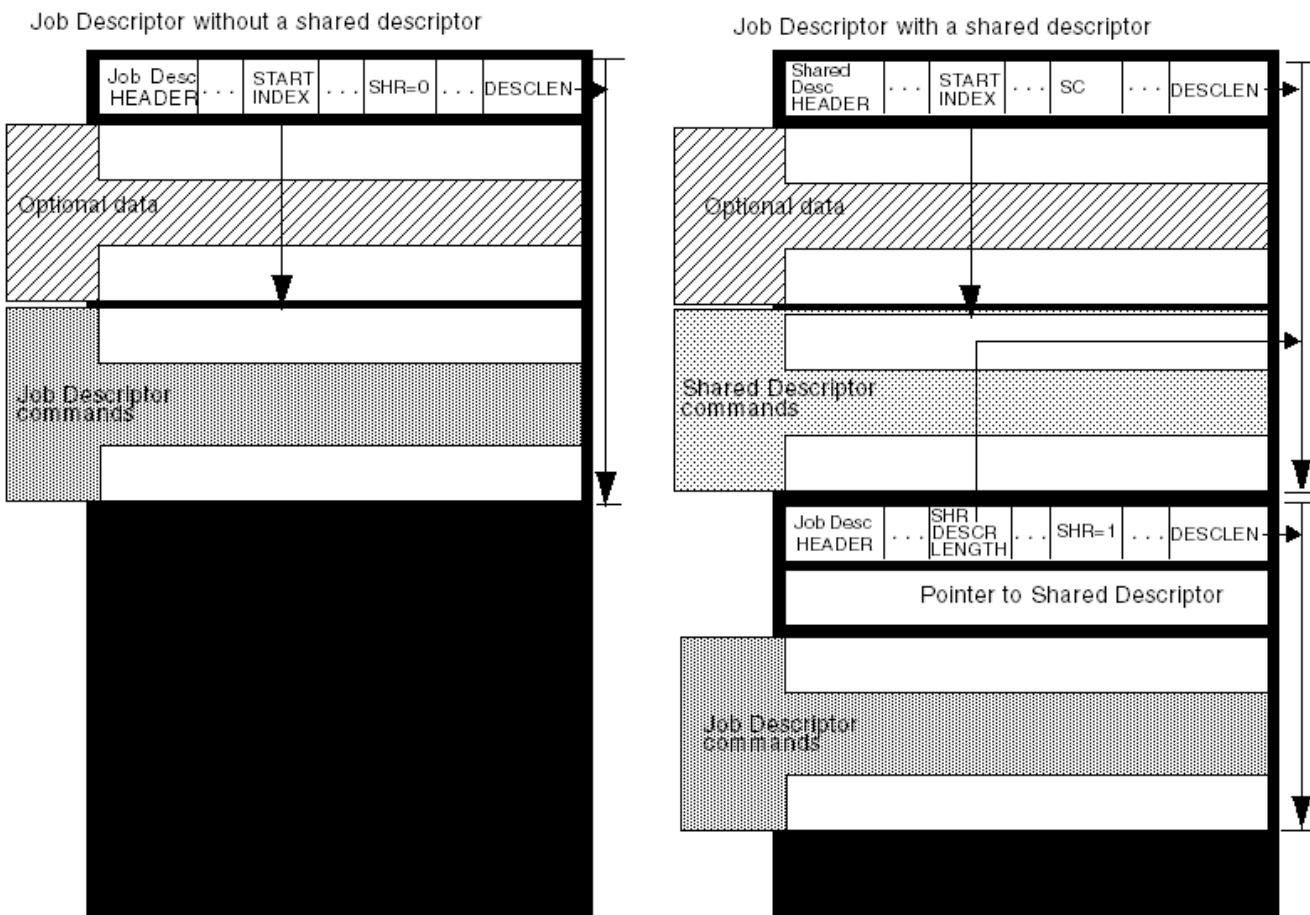


Figure 5-9. Job descriptor layout in descriptor buffer

#### 5.6.7.1.1 Executing commands when SHR = 0

When SHR = 0, the job descriptor does not reference a shared descriptor. Therefore, the HEADER's START INDEX field specifies the position of the next command that will execute within the job descriptor. If the START INDEX value is 0 or 1, the next command to execute is the command immediately following the HEADER command. Any other value causes a jump to the position indicated by the START INDEX field. Before the job descriptor continues execution, the remainder of the job descriptor is

## **Descriptors and descriptor commands**

fetched from memory and loaded into the DECO's descriptor buffer. The left half of [Figure 5-9](#) shows the layout of a job descriptor that does not reference a shared descriptor.

Commands execute in the order in which they appear in the descriptor buffer until one of the following is executed:

- The last command in the job descriptor
- A JUMP command ([see JUMP \(HALT\) command](#))
- An in-line descriptor ([see Using in-line descriptors](#))
- A replacement job descriptor ([see Using replacement job descriptors](#))

When JUMP commands are executed, the behavior is as follows:

- If DECO executes an unconditional halt type of JUMP or a conditional halt type whose tested condition evaluates to true, execution of the job descriptor terminates.
- If DECO executes a JUMP whose type is conditional halt, local conditional jump, non-local conditional jump, conditional subroutine call, or conditional subroutine return and the tested condition evaluates to false, execution continues with the next command following the JUMP.
- If the JUMP type is local or non-local jump or conditional subroutine call and the tested condition evaluates as true, the command indicated by the LOCAL OFFSET field (for local jumps) or by the Pointer Field (for non-local jumps) is the next command to execute.
  - If the jump is local, the target of the JUMP must be within the current job descriptor.
  - If the jump is non-local, the target of the JUMP must be the start of a job descriptor.
  - If the JUMP type is conditional subroutine return and the tested condition evaluates as true, the next command to execute is the command following the most recently executed conditional subroutine call (which may have been within an embedded protocol).

### **5.6.7.1.2 Executing commands when SHR = 1**

As described in [Executing commands when SHR = 0](#), the portion of the job descriptor (including the HEADER command) contained in the holding tank is loaded into the descriptor buffer. When SHR = 1, the job descriptor references a shared descriptor ([see Shared descriptors](#)).

In this case, instead of a START INDEX field, the job descriptor HEADER contains a SHR DESCR LENGTH field. This field specifies the length of the shared descriptor, which allows DECO to leave enough space for the shared descriptor when the job descriptor is loaded into the descriptor buffer. [Figure 5-9](#) shows the layout of a job descriptor that references a shared descriptor on the right.

A pointer to the shared descriptor's location in memory appears in the word immediately following the job descriptor HEADER. The pointer, together with the MID that was used when fetching the descriptor, is used to determine if the shared descriptor is already resident in the DECO, and is therefore a candidate for sharing. If the shared descriptor is not resident or cannot be shared, the shared descriptor is fetched from memory using the pointer as the starting address. Processing cannot continue until the entire shared descriptor is present. The START INDEX field within the shared descriptor's HEADER specifies the position of the next command that will execute within the shared descriptor, once the shared descriptor begins execution.

### 5.6.7.1.3 Executing commands when REO = 0

If the job descriptor references a shared descriptor, the second command to be executed depends on the REO setting. When REO = 0, DECO executes the shared descriptor before the remainder of the job descriptor, as illustrated on the left of [Figure 5-10](#). After the job descriptor HEADER executes, the HEADER command within the shared descriptor (2.0 in the diagram) is the next command to execute. The commands within the shared descriptor then execute in sequential order, except in the case of a JUMP command. If the tested condition specified by a conditional JUMP evaluates as true, the next command to execute is the jump's target command. The jump target for a local JUMP or subroutine call must be within the descriptor buffer. The jump target for a non-local JUMP must be the HEADER of a job descriptor located in memory.

Note that loading the new job descriptor into the descriptor buffer is delayed if the delay is required to satisfy any pending shared descriptor sharing requests. Once the new job descriptor begins loading, the shared descriptor must be refetched from memory for any other job descriptors to use it.

The shared descriptor terminates when any of the following is executed:

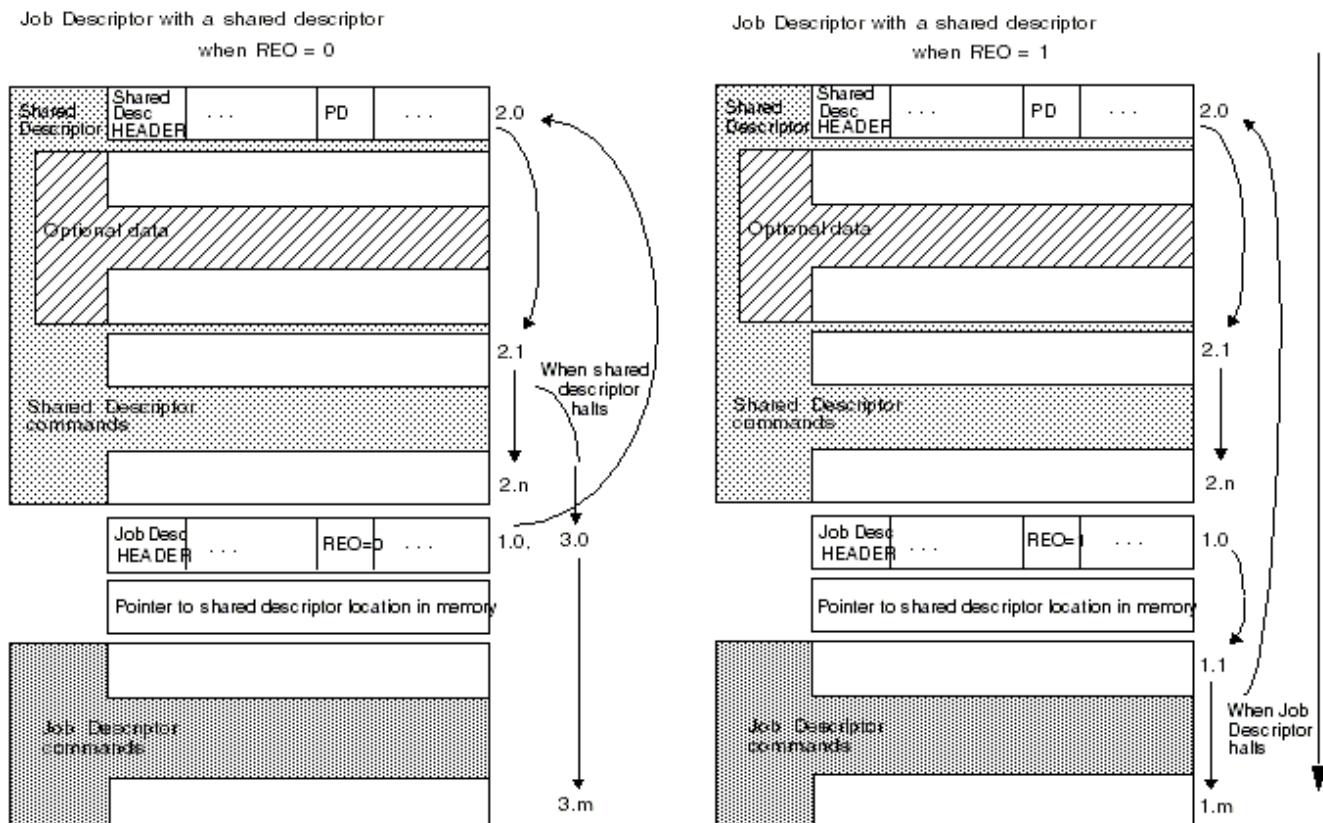
- In-line job descriptor
- Replacement job descriptor
- JUMP HALT command
- Non-local JUMP
- Subroutine call
- Subroutine return

## Descriptors and descriptor commands

Once the shared descriptor terminates, DECO then executes the job descriptor, starting with the job descriptor HEADER, which is executed a second time (3.0 in the diagram). Typically the shared descriptor terminates by executing the job descriptor HEADER command that immediately follows the shared descriptor in the descriptor buffer. This simultaneously terminates the shared descriptor and starts the job descriptor.

### 5.6.7.1.4 Executing commands when REO = 1

When REO is 1, DECO executes the job descriptor before the shared descriptor, as illustrated in the diagram on the right of [Figure 5-10](#). In this case, the job descriptor command that immediately follows the shared descriptor pointer (1.1 in the diagram) executes immediately after the job descriptor HEADER. After the job descriptor terminates, DECO then executes the shared descriptor commands, starting with the shared descriptor HEADER (2.0 in the diagram).



**Figure 5-10. Order of command execution if a shared descriptor is referenced**

### 5.6.7.1.5 Executing additional HEADER commands

A job descriptor must start with a job descriptor header, and a shared descriptor must start with a shared descriptor header. These are typically the only HEADER commands within a descriptor, but it is possible for the descriptor to have additional HEADER commands.

No error is generated if a job descriptor or shared descriptor executes additional shared descriptor HEADER commands. These are essentially no-ops, with one exception. If START INDEX is non-zero, the Shared Descriptor HEADER command causes a jump to that position within the descriptor buffer. That is, the Shared Descriptor HEADER command executes as if it is an unconditional JUMP to an absolute index. Note that this is different from the JUMP command, which uses relative addressing. The first shared descriptor HEADER command is the one that is treated as real. All subsequent shared descriptor HEADER commands executed, including the first one if executed again, are no-ops (other than an absolute jump if the START INDEX is nonzero).

#### NOTE

It is an error to execute a shared descriptor HEADER command in a job descriptor when there is no shared descriptor (when SHR= 0).

If a job descriptor does not reference a shared descriptor, any additional job descriptor HEADER commands that it executes (for example, by jumping back to the beginning of the job descriptor) are treated as jumps to an absolute address within the descriptor buffer. This address must be within the job descriptor. If a job descriptor does reference a shared descriptor, any additional job descriptor HEADER commands that it executes are treated as no-ops. Executing a job descriptor HEADER command within a shared descriptor terminates the shared descriptor if the shared descriptor runs before the job descriptor runs (i.e. REO = 0), but the job descriptor header acts as a no-op if the shared descriptor runs after the job descriptor runs (i.e. REO = 1).

### 5.6.7.1.6 Jumping to another job descriptor

Note that either a job descriptor or a shared descriptor can execute a non-local JUMP to a job descriptor. In these cases the current job descriptor or shared descriptor terminates, and the new job descriptor is fetched into the descriptor buffer and executes. Note that this new job descriptor is not permitted to reference a shared descriptor, but can also execute a non-local JUMP to another job descriptor. This mechanism allows the construction of jobs that are larger than the descriptor buffer. Once the entire chain of job descriptors terminates, a single job termination status word (see [Job termination status/error codes](#)) is written to memory. The return status is as if the original job descriptor had completed. That is, for job ring jobs, the original job descriptor address is placed in the appropriate output ring.

## 5.6.7.2 Command properties

Three properties determine how CAAM handles each command:

- Blocking
- Load/store checkpoint
- Done checkpoint

### 5.6.7.2.1 Blocking commands

A blocking command must complete before the next command can begin. Note that the completion is from the standpoint of the DECO. If the command requires a read and the DECO has scheduled the read, the next command can begin even if the read has not completed.

Most commands are blocking commands. The notable exceptions are commands that perform LOADs, STOREs and MOVEs.

### 5.6.7.2.2 Load/store checkpoint

If a command is a load/store checkpoint, it must wait for all prior LOADs and/or STOREs to complete before it can start. This property ensures that LOADs and STOREs occur in proper order. LOADs, STOREs, and information FIFO accesses can checkpoint separately.

### 5.6.7.2.3 Done checkpoint

If a command is a done checkpoint, it must wait until all current cryptographic activity associated with the descriptor is done. The CHAs signal the done once all computation has completed. Note that this is different from the descriptor being done, since not all loads and stores may have completed. It merely indicates that the CHAs in use have completed their current tasks.

## 5.6.7.3 Command types

The following is a list of commands that are supported in CAAM along with their blocking and checkpoint properties.

**Table 5-13. List of command types**

Command name	CTYPE	Blocking	Load/store checkpoint	Done checkpoint	See section/page
KEY (& SEQ KEY)	00000 (00001)	Yes	Load/Store	Yes	<a href="#">KEY commands</a>
LOAD (& SEQ LOAD)	00010 (00011)	No	Load	No	<a href="#">LOAD commands</a>
FIFO LOAD (& SEQ FIFO LOAD)	00100 (00101)	No	Load	No	<a href="#">FIFO LOAD command</a>
STORE (& SEQ STORE)	01010 (01011)	No	Store	If from a Context Register the corresponding CHA must be done	<a href="#">STORE command</a>
FIFO STORE (& SEQ FIFO STORE)	01100 (01101)	No	Store, also a Load checkpoint if encrypting	No	<a href="#">FIFO STORE command</a>
MOVE (& MOVE_LEN)	01111 (01110)	Yes, if WC set	Load or Store if CAAM DMA, depending on type of MOVE. It is a MOVE checkpoint if the CCB DMA is being used.	If from a Context Register the corresponding CHA must be done	<a href="#">MOVE and MOVE_LEN commands</a>
OPERATION (ALGORITHM OPERATION) (PROTOCOL OPERATION)	10000	Yes, if protocol		No	<a href="#">ALGORITHM OPERATION command</a> <a href="#">PROTOCOL OPERATION commands</a>
SIGNATURE	10010	Yes, when verifying or re-signing	No	N/A	<a href="#">SIGNATURE command</a>
JUMP	10100	Yes	Checkpoint based upon condition bits	If Class bit or bits are set	<a href="#">JUMP (HALT) command</a>
MATH	10101	No	Will wait for data if SRC1 is Input or Output Data FIFO	No	<a href="#">MATH command</a>
HEADER Job descriptor HEADER Shared Descriptor HEADER	10110 10111	Yes	N/A	N/A	<a href="#">HEADER command</a> <a href="#">HEADER command</a>
SEQ IN PTR	11110	Yes	No	No	<a href="#">SEQ IN PTR command</a>
SEQ OUT PTR	11111	Yes	No	No	<a href="#">SEQ OUT PTR command</a>

### 5.6.7.4 SEQ vs non-SEQ commands

CAAM can process networking protocol packets that consist of separate fields, such as headers, sequence numbers, AADs, payloads, and ICVs. (A complete discussion of network security protocol packet formats is beyond the scope of this document.) To help process such packets efficiently, CAAM provides sequence (SEQ) versions of the following descriptor commands:

- KEY
- LOAD
- STORE
- FIFO LOAD
- FIFO STORE

SEQ and non-SEQ versions of descriptor commands have identical functions, with the major distinction being that the SEQ versions do not require pointers, because CAAM instead uses sequence addresses that were defined by previously executed SEQ IN PTR or SEQ OUT PTR commands. Because SEQ commands use previously defined addresses, SEQ commands (with the exception of SEQ STORE) do not have immediate data modes.

#### 5.6.7.4.1 Creating a sequence

Sequences are generally associated with shared descriptors (see [Shared descriptors](#)), which support a one-time definition of a set of commands to be performed on each packet in a flow. The address and length of the input and output packets is usually specified in a job descriptor (see [Job descriptors](#)) that references a shared descriptor containing SEQ-version commands to indicate how to process the data. The shared descriptor is analogous to a subroutine, and the job descriptor is analogous to a software program supplying arguments and then calling that subroutine.

The job descriptor uses the following commands to provide information about the data to be processed by the sequence

- A SEQ IN PTR command to specify the length and address of the data to be processed by the sequence (see [SEQ IN PTR command](#)).
- A SEQ OUT PTR command to specify the length and address of the buffer for the sequence output data (see [SEQ OUT PTR command](#)).

The SEQ IN PTR and SEQ OUT PTR commands have an SGF field, which when set to 1, allows sequence input and output areas to be defined by means of scatter/gather tables.

Once the input and/or output sequence pointers have been set, subsequent SEQ commands indicate how to process the packet. The length of the sequence may be extended by issuing additional SEQ IN PTR and SEQ OUT PTR commands with the PRE bit set (see [Table 5-63](#) and [Table 5-65](#)). DECO tracks how far into the output sequence DECO has progressed, and this information is used if a rewind is needed so that a second pass can be made over the sequence (see RTO field in SEQ IN PTR and SEQ OUT PTR commands).

An input sequence ends when any of the following occurs:

- All specified input data is consumed.
- A new input sequence is started.
- An error occurs.

There can be at most one scatter/gather table active for input and at most one scatter/gather table active for output in the DECO at any time. Note that non-sequential commands can be executed within the same descriptor while a sequence is running. An input gather table can be in use by either an input sequence or by non-SEQ KEY LOAD or FIFO commands, but not both. Likewise, an output scatter table can be in use by either an output sequence or by non-SEQ STORE or FIFO STORE commands, but not both.

To accelerate performance, CAAM caches gather table and scatter table entries in registers (see the Gather Table Register (DxGTR) and Scatter Table Register (DxSTR)).

#### **NOTE**

If a scatter/gather table is being used for an input or output sequence, and a non-SEQ command references a second scatter/gather table for input or output data, entries from the second scatter/gather table overwrite the entries from the initial scatter/gather table. This can result in the input/output sequence referencing the wrong data.

#### **NOTE**

Hardware does not flag this as an error. The descriptor programmer must ensure this does not happen.

#### **5.6.7.4.2 Using sequences for fixed and variable length data**

Some SEQ commands act on fixed length data (for example, keys, IVs, or packet header fields) whereas other SEQ commands act on data that changes from packet to packet, such as packet payload. The VLF bit found in most SEQ commands indicates whether the data associated with the SEQ command is a constant length or whether the length is to be found in the corresponding variable length registers (VSIL and VSOL).

Note that the VSIL and VSOL are not accessible through the register bus, but rather they are read from or written to by means of the MATH Command (see [Table 5-60](#)). This allows commands within the descriptor to calculate the length of the input sequence. Given a total packet length and start pointer (from an internal or externally generated job descriptor), the descriptor can calculate the variable length portion of a job and load it into the Variable Length Registers to be referenced by subsequent SEQ commands (by setting the VLF bit). For jobs submitted by means of the job ring interface, each job ring can be individually configured via the INCL\_SEQ\_OUT field in the Job Ring Configuration Register to output an additional word in the Output Ring that indicates the length of the output sequence (that is, the number of bytes output via SEQ commands).

#### **5.6.7.4.3 Transferring meta data**

When processing data, CAAM typically uses DMA to read input data and write output data. Because CAAM is primarily intended to accelerate cryptographic operations, the output data is normally different from the input data. However, it is possible to use CAAM's external DMA to transfer data from an input buffer to an output buffer without modifying the data (i.e. the identity transformation, also called null encryption), typically to either:

- Benefit from CAAM's scatter/gather capabilities
- Transfer meta data in conjunction with cryptographic processing

The latter case is often useful because the meta data may describe the type, the source, the destination, the classification, the priority and/or the amount of the cryptographic data. If this meta data appears ahead of the cryptographically processed data, it is called 'leading meta data'. If it appears after, it is called 'trailing meta data'.

Meta data can be transferred from input to output without modification in the following ways:

- Use the FIFO LOAD and FIFO STORE commands; if the FIFO STORE command specifies output data type 30 (message data), the command transfers data without transformation.
- Define an input sequence with a SEQ IN PTR command and an output sequence with a SEQ OUT PTR command; then use the SEQ FIFO LOAD and SEQ FIFO STORE commands to initiate the transfer.
- Use FIFO LOAD and FIFO STORE commands to transfer either the meta data or the cryptographically processed data and the SEQ FIFO LOAD/SEQ FIFO STORE commands to transfer the other type of data.

If using the second method above, be aware that this method transfers all data in the sequence from input to output without any transformation. If some data must be transferred as untransformed meta data and some as cryptographically processed data, either use separate sequences to transfer the different data types or use one of the other methods.

#### 5.6.7.4.4 Rewinding a sequence

Note that it is possible to rewind a sequence to make an additional pass over the input and output data (see RTO field in SEQ IN PTR and SEQ OUT PTR commands). A rewind can fill in data that was skipped over in a previous pass. For example, a rewind may be necessary if a field contains a hash value that is computed over data that appears later in the output PDU.

#### 5.6.7.5 Information FIFO entries

The DECO has an information FIFO (NFIFO) as a companion to the input data FIFO. The information FIFO holds entries that describe the corresponding data in the input data FIFO and provide a means for generating padding and random numbers. The info FIFO also describes data to be obtained from the output FIFO and from the auxiliary data FIFO.

Typically, using a FIFO command to load data into the input data FIFO causes CAAM to automatically generate the proper information FIFO entries to handle that data. However, that functionality can be overridden to allow the descriptor to directly specify the information FIFO entries. Entries can be placed into the information FIFO via a LOAD Immediate command with a DST value of 7Ah.

#### 5.6.7.6 Cryptographic class

CAAM divides cryptographic algorithms into two different classes for the purpose of selecting CHAs. Some key and data movement commands must have a CLASS value associated with them so they are delivered to the proper CHA.

The following table shows the class for each cryptographic algorithm.

**Table 5-14. Classes of the cryptographic algorithms**

Class	Algorithm
Class 1	AES (all modes), DES, 3DES, ARC4 , RNG

*Table continues on the next page...*

**Table 5-14. Classes of the cryptographic algorithms (continued)**

Class	Algorithm
Class 2	MD5, SHA-1, SHA-224, SHA-256

**NOTE**

A descriptor that requests both a Class 1 CHA and a Class 2 CHA must request the Class 2 CHA first. Otherwise, in versions of CAAM that implement more than one DECO a deadlock situation could occur as follows:

- Descriptor  $x$  executing in DECO  $x$  acquires CHA 1 and then requests CHA 2.
- Descriptor  $y$  in DECO  $y$  acquires CHA 2 and then requests CHA 1.
- Descriptor  $x$  and descriptor  $y$  wait until both CHAs are available, but neither will release the CHA that it currently has.

If descriptors always acquire CHAs in the same order, this deadlock situation is avoided. The required order is Class 2 first, then Class 1. An error is generated if a Class 2 CHA is selected after a Class 1 CHA.

Note that software written for versions of CAAM that implement only one DECO must still follow this practice to ensure that the software is portable to versions of CAAM that implement two or more DECOs.

When specifying classes in commands, a two-bit field is used to specify class as follows:

**Table 5-15. Class field**

Class Value	Meaning for LOAD and STORE Commands	Meaning for Other Commands
00	CCB class independent	None, sequential data skipped for SEQ FIFO LOAD command.
01	CCB Class 1	Class 1
10	CCB Class 2	Class 2
11	DECO	Both Class 1 and Class 2

### 5.6.7.7 Address pointers

Many of the descriptor commands and several data structures used by CAAM include Pointer fields. All address pointers used by CAAM are 32-bits in length, as shown in the table below.

**Table 5-16. Format of 32-bit pointers**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32-bit address																															

### 5.6.8 HEADER command

Every descriptor begins with a HEADER command, which provides basic information about the descriptor itself, such as length, ability of DECOs to share the descriptor, and whether errors are propagated when the descriptor is shared. The HEADER command does not provide any data movement or processing information.

Job descriptor and shared descriptor HEADER commands share a base format, but some fields are specific to descriptor type. The Job Descriptor Header Command format is shown in [Table 5-17](#), the Shared Descriptor Header Command format in [Table 5-18](#), and the fields of both are described in detail in [Table 5-19](#).

**Table 5-17. Job descriptor header command format**

31–27					26	25	24	23	22	21–16
CTYPE = 10110					0	RSMS	DNR	ONE	0	START INDEX / SHR DESCRL LENGTH
15	14	13	12	11	10–8		724	6–0		
ZRO	TD	MTD	SHR	REO	SHARE		0	DESCLEN		
If SHR = 1, a one word shared descriptor pointer is located immediately after the HEADER.										

**Table 5-18. Shared descriptor header format**

31–27					26	25	24	23	22	21–16
CTYPE = 10111					0	RIF	DNR	ONE	0	START INDEX
15	14	13	12	11	10	9–8		724	6	5–0
ZRO	0	CIF	SC	PD	0	SHARE		0	0	DESCLEN

**Table 5-19. Descriptor header field descriptions**

Job descriptor header fields	Shared descriptor header fields	Description
31-27 CTYPE	31-27 CTYPE	Command type 10110 Job descriptor 10111 Shared descriptor
26	26	Reserved
25 RSMS	25 RIF	<p><b>In Job Header:</b> RSMS</p> <p>Requires SEQ MID to be the same. This bit is used to ensure that MID-based access control cannot be bypassed by sharing shared descriptors. Setting RSMS = 1 in a job descriptor header prevents another job descriptor from obtaining keys or other shared descriptor data by sharing them from the first job descriptor if the second job descriptor is not using a MID value that enabled it to read the shared descriptor directly from memory.</p> <p>0 The MIDs of two job descriptors do not have to match in order for them to share the same shared descriptor.</p> <p>1 The MIDs of two job descriptors must match in order for them to share the same shared descriptor. If they do not match, an error is reported.</p> <p><b>In Shared Header:</b> Read Input Frame</p> <p>As soon as possible, DECO should read the entire input frame as defined in a SEQ IN PTR command in the job descriptor. The length of the input frame is placed in the VSIL Register.</p>
24 DNR	24 DNR	<p>Do Not Run</p> <p>0 Normal execution</p> <p>1 Do Not Run. There was a problem upstream so this descriptor should not be executed.</p> <p><b>NOTE:</b> If this bit is found in a job descriptor header, CAAM still fetches any associated shared descriptor. If the shared descriptor header's PD bit is set and the DNR bit is not set, CAAM updates the shared descriptor header's DNR bit. As a result, future job descriptors that use this shared descriptor do not run. Once software clears the DNR bit in the shared descriptor, any new job descriptors that use this shared descriptor run normally.</p>
23 ONE	23 ONE	<p>One</p> <p>The ONE bit is always 1. This bit is used in combination with the ZRO bit to verify that the endianness of the header is correct. This is necessary because CAAM is used in chips with both big-endian and little-endian processors.</p>
22 Reserved	22 Reserved	Reserved
21-16 START INDEX/ SHR DESCRIPTOR LENGTH	21-16 START INDEX	<p><b>In Job Header:</b></p> <ul style="list-style-type: none"> <li>If SHR = 0, this is the START INDEX field, which specifies the index of the word in the descriptor buffer where execution of the job descriptor should start. That is, DECO should jump to the specified word to start processing.</li> <li>If SHR = 1, this is SHR DESCRIPTOR LENGTH field, which specifies the length of the Shared Descriptor (in 32-bit words).</li> </ul> <p><b>In Shared Header:</b> This is the START INDEX field, which specifies the index of the word in the descriptor buffer where execution of the shared descriptor should start. This allows protocol or other information to be jumped over.</p>

Table continues on the next page...

**Table 5-19. Descriptor header field descriptions (continued)**

Job descriptor header fields	Shared descriptor header fields	Description
15 ZRO	15 ZRO	Zero  The ZRO bit is always 0. This bit is used in combination with the ONE bit to verify that the endianness of the header is correct. This is necessary as CAAM is used in Freescale product lines with both big endian and little endian processors.
14 TD	14 RSV	<b>In Job Header:</b> Trusted Descriptor <ul style="list-style-type: none"> <li>• If TD = 0, this is a normal job descriptor.</li> <li>• If TD = 1, this is a trusted descriptor.</li> </ul> See the discussion <a href="#">Trusted descriptors</a> concerning trusted descriptors.  <b>In Shared Header:</b> Reserved
13 MTD	13 CIF	<b>In Job Header:</b> Make trusted descriptor.  See the discussion <a href="#">Trusted descriptors</a> about trusted descriptors.  <b>In Shared Header:</b> Clear Input FIFO (CIF)  If set, the input info FIFO entries are reset between self-shared descriptors. That is, these are reset if the next job to be run within a DECO executes the same shared descriptor as the previous job run in that same DECO. (The Input and iNfo FIFO are always reset between descriptors that don't share the same shared descriptor.)
12 SHR	12 SC	<b>In Job Header:</b> Shared Descriptor (SHR) flag  If set, this descriptor has a shared descriptor that is pointed to by the next word or words. SHR controls how START INDEX / SHR DESCRIPTOR LENGTH is used.  <b>In Shared Header:</b> Save Context (SC)  When a shared descriptor is shared between two successive jobs run in the same DECO (self-sharing): <ul style="list-style-type: none"> <li>0 The context is erased.</li> <li>1 The context is saved.</li> </ul>
11 REO	11 PD	<b>In Job Header:</b> Reverse Execution Order (REO)  This bit is ignored if SHR = 0 (i.e. no shared descriptor). Normally, the shared descriptor is executed prior to the job descriptor. Setting this bit reverses that order. Setting the REO bit in a trusted descriptor results in an error indication.  <b>In Shared Header:</b> Propagate DNR (PD)  If the job descriptor's DNR bit is set and this bit is set, set the DNR bit of the shared descriptor header if it is not already set.
10-8 SHARE	10 RSV	<b>In Job Header:</b> All three bits (10-8 ) are part of the SHARE field. See explanation below.  <b>In Shared Header:</b> Bit 10 is reserved.
	9-8 SHARE	In both Job Header and Shared Header: Share State (SHARE)  This defines if, and when, the shared descriptor of this descriptor can be shared with that of another (See <a href="#">Table 5-10.</a> ) Also see <a href="#">Specifying different types of shared descriptor sharing</a> " for further information.
7 RSV	7 RSV	Reserved

*Table continues on the next page...*

**Table 5-19. Descriptor header field descriptions (continued)**

Job descriptor header fields	Shared descriptor header fields	Description
6-0 DESCLEN	6	<b>In Job Header:</b> All seven bits are part of the DESCLEN field. See explanation below. <b>Shared Header:</b> Bit 6 is reserved.
	RSV	
5-0 DESCLEN	5-0	In both Job Header and Shared Header: Descriptor Length
	DESCLEN	This field represents the total length in 4-byte words of the descriptor. A descriptor length of 0 is undefined. The header word is included in the length. Note that the size of the descriptor buffer is 64 words, so that is the maximum size of a single descriptor.

If the SHR bit in a job descriptor header command is set, a pointer to the shared descriptor immediately follows the header. The shared descriptor address is in a single word. See [Address pointers](#).

## 5.6.9 KEY commands

### NOTE

In the following discussion, the term 'KEY command' refers to both the SEQ and non-SEQ forms of the command.

KEY commands are used to load keys into one of CAAM's key registers: Class 1 or Class 2 Key Register, AFHA S-box. The SEQ KEY command is identical to the KEY command except that no address is specified and the immediate bit cannot be set. See [SEQ vs non-SEQ commands](#).

If the key to be loaded into a key register is encrypted (see [Black keys](#)), CAAM can be told to automatically decrypt it as it is loaded into the key register. The MOVE command, LOAD command and KEY command can all be used to load a Red Key into a key register, but only the KEY command can be used to load a Black Key. Note that Black Keys can be loaded only into key registers because only the KEY and SEQ KEY commands decrypt Black Keys, and the key registers are the only possible destinations for these commands.

If the KEY command is loading a Black Key, the Class 2 key must be loaded prior to the Class 1 key as the loading of encrypted keys has side effects on the Class 1 Key Register. (See [Cryptographic class](#) for a discussion concerning the order of requesting CHAs).

If ENC is set (i.e. a Black Key is being loaded), the KEY command has significant side effects, including clearing the following:

- Input Data FIFO

- Output Data FIFO
- Class 1 Key Register
- Class 1 Data Size Registers
- Class 1 Mode Register
- Class 1 Context (if EKT is also set)

As a result, the only commands that should precede loading a Black Key are:

- JUMP
- SEQ IN PTR
- SEQ OUT PTR
- LOADs to registers not mentioned above.
- MOVEs to or from registers not mentioned above.

### NOTE

To preserve the secrecy of key material, the KEY command is blocking until the key is safely loaded. The KEY command does not start until all owned CHAs have completed their operations, and the DECO hangs if the CHA cannot finish until after the KEY command finishes.

**Table 5-20. KEY command format**

31–27					26–25		24	23	22	21	20	19–16							
CTYPE = 00000 or 00001					CLASS		SGF	IMM	ENC	NWB	EKT	KDEST							
15	14	13	12	11	10	9–0													
TK	0	0	0	0	0	LENGTH													
31–16																			
Pointer (one word, see <a href="#">Address pointers</a> ) (or Value if immediate, one or more words)																			
15–0																			
Pointer (or Value if immediate, one or more words)																			

**Table 5-21. KEY command field descriptions**

Field	Description
31–27 CTYPE	Command Type 00000 KEY 00001 SEQ KEY

*Table continues on the next page...*

**Table 5-21. KEY command field descriptions (continued)**

Field	Description
26-25 CLASS	Class. This defines whether this key is for Class 1 or Class 2. 00 Reserved 01 Class 1 Key 10 Class 2 Key 11 Illegal. Reserved. <b>NOTE:</b> Class must be set to Class 1 if the key DEST field is set to 01 or 10.
24 SGF	Scatter/Gather Table Flag (SGF) If CTYPE = 00000 (KEY), this bit is the scatter/gather table flag. 0 Pointer points to actual data. 1 Pointer points to a scatter/gather table. <b>NOTE:</b> It is an error for this bit to be set if the IMM bit is set. It is also an error for this bit to be set when reading a key from a Secure Memory key partition (i.e. a partition whose SMAP and SMAG register settings do not permit an ordinary read transaction). If CTYPE = 00001 (SEQ KEY), setting this bit is an error.
23 IMM	Immediate Flag 0 The key value is found at the location pointed to in the next word. 1 The key value follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. <b>NOTE:</b> An AFHA SBOX cannot be supplied as immediate data because it is too large to fit in the descriptor buffer. If the SGF bit is set, It is an error for the IMM bit to be set.
22 ENC	Key is encrypted If ENC = 0, the key is assumed to be in plaintext and is loaded into the destination register without decryption. If ENC = 1, CAAM automatically decrypts the key (using the JDKEK, or if this is a trusted descriptor, using the TDKEK if TK = 1) before putting it in the key register. Decrypting a key requires using the AESA, Class 1 Mode register, Key Size, and key registers. Therefore, Class 2 Black Keys (if any) must be loaded prior to loading any of the Class 1 registers, and Class 1 Black Keys must be loaded prior to loading any other Class 1 registers. Setting this bit sets the Key register's ENC flag (see <a href="#">Figure 5-12</a> ).
21 NWB	No Write Back 0 Allows the key that is loaded into the key register to be written back out to memory (as a Black Key) with the FIFO STORE command. 1 Prevents the key that is loaded into the key register from being written back out to memory. NWB applies to all key registers: Class 1 Key Register, Class 2 Key Register, and AFHA S-Box. Setting this bit sets the key register's NWB flag (see <a href="#">Figure 5-12</a> ). The No Write Back setting lasts until the end of the descriptor (or sequence of shared descriptors) or until the corresponding key register or CHA is cleared/reset.

*Table continues on the next page...*

**Table 5-21. KEY command field descriptions (continued)**

Field	Description
20 EKT	Encrypted Key Type  This field is ignored if ENC = 0. The EKT bit determines which decryption mode is used when a Black Key (ENC = 1) is loaded.  0 AES-ECB 1 AES-CCM  A Black Key encrypted with AES-ECB must be decrypted with AES_ECB, and a Black Key encrypted with AES-CCM must be decrypted with AES_CCM. If the wrong mode is selected, no error is issued. However, the value loaded into the key register will be incorrect. See <a href="#">Black keys</a> for more information.  Setting this bit sets the Key Register's EKT flag (see <a href="#">Figure 5-12</a> ).
19-16 KDEST	Key Destination  0000 Key Register (as per the CLASS field)  0010 AFHA SBOX. This key destination requires CLASS = 01 (Class 1 key).  0011 MDHA Split Key. An MDHA split key is the concatenation of the IPAD followed by the OPAD. Split keys offer higher performance for HMACs. Note that MDHA IPAD/OPAD values are considered keys and are decrypted to the Class 2 Key Register. This key destination requires CLASS = 10 (Class 2 key).  All other values are reserved.
15 TK	Trusted Key  This bit is used only by trusted descriptors. If not a trusted descriptor, setting TK = 1 and ENC = 1 is an error. If the ENC bit is not set, this bit is ignored.  0 The trusted descriptor wants to use the Job Descriptor Key Encryption Key (JDKEK) to decrypt the key to be loaded into a key register. 1 The trusted descriptor wants to use the Trusted Descriptor Key Encryption Key (TDKEK) to decrypt the key to be loaded into a key register.
14-10	Reserved
9-0 LENGTH	Key Length  This field defines the length of the key in bytes. If the key is encrypted, this is the decrypted length of the key material only. The built-in key decryption operation produces output whose length is as specified in the LENGTH field. ECB encrypted keys are padded to 16-byte boundaries, so the KEY command reads enough input to read the entire encrypted key. CCM-encrypted keys have a 6-byte nonce, a 6-byte MAC, and padding of up to 7 bytes. The length is checked to ensure it is not too large for the specified destination. It is an error if LENGTH is less than 16 when reading a key from a Secure Memory key partition (meaning a partition whose SMAP register settings do not permit an ordinary read transaction). If the destination is the AFHA S-box, the length must be exactly 258 bytes.

## 5.6.10 LOAD commands

### NOTE

In the following discussion, the term 'LOAD command' refers to both the SEQ and non-SEQ forms of the command.

## Descriptors and descriptor commands

LOAD commands are used to load values into registers, either directly from the descriptor (a LOAD IMMEDIATE command contains constant data within the command) or from a memory location addressed by a pointer within the command. The SEQ LOAD command is identical to the LOAD command except that no address is specified and the immediate bit cannot be set. See [SEQ vs non-SEQ commands](#).

The definitions of the OFFSET and LENGTH fields in the LOAD command can depend on the CLASS and destination (DST) fields. The first table shows the command fields, and the second table defines OFFSET and LENGTH as well as additional behaviors of the command.

**Table 5-22. LOAD command format**

31–27	26	25	24	23	22–16
CTYPE = 00010 or 00011	CLASS	SGF	IMM	DST	
<b>15–8</b>					<b>7–0</b>
OFFSET					LENGTH
<b>31–16</b>					
Pointer (one word, see <a href="#">Address pointers</a> ) (or Value if immediate, one or more words)					
<b>15–0</b>					
Pointer (or Value if immediate, one or more words)					

**Table 5-23. LOAD command field descriptions**

Field	Description
31–27 CTYPE	Command Type 00010 LOAD 00011 SEQ LOAD
26–25 CLASS	Class 00 Load class-independent objects in CCB 01 Load Class 1 objects in CCB 10 Load Class 2 objects in CCB 11 Load DECO objects
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Field (VLF) flag. Meaning depends on CTYPE. If CTYPE = 00010 (LOAD), this bit is the scatter/gather table flag. 0 Pointer points to actual data. 1 Pointer points to a scatter/gather table. <b>NOTE:</b> If the IMM bit is set, it is an error for this bit to be set. If CTYPE = 00011 (SEQ LOAD), this bit is the Variable Length Field flag. 0 LENGTH is length of actual data. 1 Length is variable. CAAM uses the length in the Variable Sequence In Length register and ignores the LENGTH field.

*Table continues on the next page...*

**Table 5-23. LOAD command field descriptions (continued)**

Field	Description
23 IMM	<p>Immediate Flag</p> <p>If CTYPE = 00010 (LOAD)</p> <p>0 Data to be loaded is found at the location pointed to by the next word. If the immediate data is not a multiple of 4 bytes, it must be padded to the next 4-byte boundary.</p> <p>1 Data to be loaded follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word.</p> <p><b>NOTE:</b> If the SGF bit is set, it is an error for this bit to be set.</p> <p>If CTYPE = 00011 (SEQ LOAD)</p> <p>0 Mandatory.</p> <p>1 Setting this bit is an automatic error.</p>
22-16 DST	The DST value defines the destination register, such as CONTEXT, ICV, or IV. See <a href="#">Table 5-24</a> for a list of supported destinations.
15-8 OFFSET	OFFSET defines the start point for writing within the destination. The destination DST determines whether the length is specified in bytes or words. See <a href="#">Table 5-24</a> for details.
7-0 LENGTH	Length of the data. The value in the DST field determines whether the length is specified in bytes or words. See <a href="#">Table 5-24</a> for details.
31-0 POINTER/ VALUE	<p>32-bit pointer or the immediate value if IMM = 1. Note that the immediate value occupies as many words as required to fit the number of bytes specified in the LENGTH field. Data is left aligned.</p> <p><b>NOTE:</b> This field is present only for LOAD Commands (i.e. not for SEQ LOAD Commands).</p>

CAAM can accomplish the data transfer associated with a LOAD immediate command in three different ways:

- Using a direct (non-DMA) path to the register, referred to as a direct immediate load
- Using CAAM's internal-transfer DMA
- Using CAAM's external-transfer DMA but transferring data within CAAM rather than by means of the external AXI bus

CAAM automatically selects the appropriate transfer mechanism as follows:

- CAAM selects the direct immediate load data path (the first bullet above) if the restrictions are met because this is the fastest of the three transfer mechanisms (see following paragraph).
- If the data length or offset restrictions are not met, but the data is word-aligned, CAAM automatically selects the internal-transfer DMA data path (the second bullet above) because this is the second fastest of the three transfer mechanisms.
- If the data is not word-aligned, CAAM loads the register by means of the external DMA (the third bullet above), which may compete with other external DMA transfers and, even if there is no competition, takes longer than the previous two transfer mechanisms.

## Descriptors and descriptor commands

The direct immediate load is the most efficient of the three transfer mechanisms, but it has the following restrictions:

- It can transfer only 4 or 8 bytes.
- The sum of the data length and the offset cannot be larger than 8, meaning the legal combinations of length and offset are either
  - 4 bytes with an offset of 0 or 4
  - 8 bytes with an offset of 0
- The data must be word-aligned.

As shown in [Table 5-24](#), some registers can be loaded only with a LOAD IMM command. These registers always use the direct immediate load data path. Other registers can be loaded using either the LOAD or LOAD IMM form of the command.

As shown in [Table 5-24](#), some LOAD destinations are control data registers and other destinations are message data registers. Data loaded into or stored from control data registers is regarded as word-oriented data, whereas data loaded into or stored from message data registers is regarded as byte-strings.

To facilitate operation in chips with different endianness configurations, byte-swapping and half-word swapping for control data registers can be configured independently from the swapping for message data registers. Also, the swapping can be configured independently for each job ring (see the Job Ring Configuration Register (JRCFGR)).

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values**

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
01	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	KEYS1	Class 1 Key Size Register	The key size registers are normally written with the KEY command. Once KEY SIZE is written, the user cannot modify the key or key size until the key is cleared.
	10				KEYS2	Class 2 Key Size Register	
02	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	DATAS1	Class 1 Data Size Register	The data size registers are normally written by means of the FIFO LOAD command.
	10				DATAS2	Class 2 Data Size Register	
03	01	Control	4/0 bytes	Yes	C1ICVS	Class 1 ICV Size Register	The ICV size registers are normally written by means of the FIFO LOAD command.
	10				C2ICVS	Class 2 ICV Size Register	

Table continues on the next page...

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values (continued)**

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
04	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	MSR	DECO MID Status Register	Consists of the following 16-bit chunks: Non-SEQ MID, SEQ MID, Reserved
06	00	Control	4/0 bytes	Yes	CCTRL	CHA Control Register	-
	11	Control	See below	Yes	DCTRL	DECO Control Register	See notes below.  The DECO Control Register is used to control the operation of DECO by means of a 1-word command that uses the LOAD command fields that normally represent OFFSET and LENGTH. This LOAD must be IMMEDIATE, which means that this DEST cannot be used with SEQ LOAD. The OFFSET and LENGTH fields are redefined as follows:  OFFSET[7:6]: MID Select for non-sequence operations (KEY, LOAD, FIFO STORE) 00 no change 01 SEQ MID 10 Non-SEQ MID (default) 11 Reserved  OFFSET[5:4]: MID Select for sequence operations (SEQ KEY, SEQ LOAD, SEQ FIFO STORE) 00 no change 01 SEQ MID (default) 10 non-SEQ MID 11 Reserved  OFFSET[3]: Disable Automatic iNformation FIFO Entries (If disable and enable are both set, disable dominates) 00 no change 01 NEVER share 10 OK to share, do propagate errors 11 OK to share, don't propagate errors  LENGTH[7]: Turn On Output Sequence Length Counting (turned off by doing sequence output pointer rewind) LENGTH[6]: Reset CHA pointer in Output Data FIFO LENGTH[5]: Reset Output Data FIFO LENGTH[4]: Process the Output Data FIFO Offset Field (automatically stalls if write burster is busy) LENGTH[3]: Reserved LENGTH[2:0]: Output Data FIFO Offset

*Table continues on the next page...*

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values (continued)**

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
07	00	Control	4/0 bytes	Yes	ICTRL	IRQ Control Register	-
08	00	Control	4/0 bytes	Yes	CLRW	Clear Written Register	-
	11	Control	0-32/ (32-length) bytes	No	DMATH0	DECO Math Register 0	Note that using the LOAD command to place values in the math registers does not update the MATH status bits (see MNV, MN, MC and MZ). Because the four math registers are in contiguous addresses, it is possible to load more than one math register simultaneously. An error is generated if the sum of the data length and the offset added to the register address causes the data to be written past the end of Math Register 3.
09	11	Control	0-24/ (24-length) bytes	No	DMATH1	DECO Math Register 1	
0A	11	Control	0-16/ (16-length) bytes	No	DMATH2	DECO Math Register 2	
0B	11	Control	0-8/ (8-length) bytes	No	DMATH3	DECO Math Register 3	Note that the values in the LENGTH/OFFSET column at the left reflect this limit.
	01	Control	4/0 bytes	Yes	AADSZ	AAD Size Register	-
20	01	Message	0-128/ (128-length) bytes	No	CTX1	Class 1 Context Register	LOAD IMM to a context register block if there are any outstanding external loads to either context register.
	10	Message	0-128/ (128-length) bytes	No	CTX2	Class 2 Context Register	A non_IMM LOAD to a context register block if the CCB DMA is writing to either context register.

Table continues on the next page...

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values (continued)**

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
40	01	Message	0-32/ (32-length) bytes	No	KEY1	Class 1 Key Register	The sum of the LENGTH and OFFSET fields must not exceed the length of the key register. The key registers are normally written by the KEY Command, but can be written by a LOAD Command using this DST value. In this case the KEY SIZE register must be written by a separate command after the KEY register has been loaded.
	10	Message	0-128/ (128-length) bytes	No	KEY2	Class 2 Key Register	LOAD IMM to a key register block if there are any outstanding external loads to either key register.
	11	Control	1-64/ offset* words	No	DESC BUF	DECO descriptor buffer	A non_IMM LOAD to a key register block if the CCB DMA is writing to either key register.  Cannot use LOAD IMM. See additional comments below.
For LOADs into the Descriptor Buffer the values in the LENGTH and OFFSET field are specified in 4-byte words.  offset* The sum of the LENGTH and OFFSET fields must be less than or equal to 64. The OFFSET is used to specify the starting word of the destination within the descriptor buffer. Note that the OFFSET is relative to the start of the job descriptor (or trusted descriptor) (which is not be the start of the descriptor buffer if there is a shared descriptor). See <a href="#">Figure 5-9</a> . For SEQ LOAD, the data written into the descriptor buffer is read from the current location pointed to by the input sequence pointer (there is no offset with respect to the source address).							
70	00	Control	4 or 8 / 0 bytes	Yes	NFSL	information FIFO and size register(s)	Using the Immediate data, write an iNfo FIFO entry and load the Size register from the DL or PL field in that iNfo FIFO entry. Also see note below.
-	-	-	The iNfo FIFO is normally written via the FIFO LOAD Command, but an entry can be written into the NFIFO using a LOAD Immediate Command with the 70h or 71h DST value. This creates an info FIFO entry from 4 or 8 bytes of IMM data and also writes to one or more size registers. The entry's DL or PL field is filled in with the same value loaded into the size register(s). The note below the row for DST 75h specifies which size registers are loaded.				

Table continues on the next page...

Descriptors and descriptor commands

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values (continued)**

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
71	00	Control	0-3/ 0 bytes	Yes	NFSM	information FIFO and size register(s)	Using the Immediate data write an info FIFO entry and load the size register from the least-significant 32-bits of the MATH register selected by means of the two ls bits of the LENGTH field. Also see note above.
72	00	Control	4 or 8 / 0 bytes	Yes	NFL	Information FIFO	Using the Immediate data write an info FIFO entry. Also see note below.
-	-	-	The info FIFO is normally written by means of the FIFO LOAD command. However, an entry can be written into the NFIFO using a LOAD Immediate command with the 72h or 73h DST value. This creates an info FIFO entry from 4 or 8 bytes of IMM data. No size registers are written.				
73	00	Control	0-3/ 0 bytes	Yes	NFM	Information FIFO	Using the Immediate data write an info FIFO entry filling in the DL or PL field from the least-significant 32-bits of the MATH register selected via the two ls bits of the LENGTH field. Also see note above.
74	00	Control	4 or 8 / 0 bytes	Yes	SL	Size register(s)	With the Immediate data in info FIFO entry format, load the size register(s) from the DL or PL field in the Immediate data but do not write an entry into the info FIFO. The note below the row for 75h specifies which size registers are loaded.
75	00	Control	0-3/ 0 bytes	Yes	SM	Size register(s)	Load the size register(s) from the value in the MATH register that is selected by means of the two ls bits of the LENGTH field. The note below the row for 75h specifies which size registers are loaded.
<b>NOTE:</b> For DST values 70h, 71h, 74h, and 75h, the particular size registers that are loaded depend on the CHAs that are selected and the DTYPE field of the entry that is written into the info FIFO.							
76	00	Message	4/0 bytes 8/0 bytes 4/4 bytes	Yes	IDFNS	Input Data FIFO Nibble Shift Register	See notes below.
			Inserts the rightmost 4 bits of the immediate value into the input to the Class 1 Alignment Block, which causes the remainder of the input data to be shifted by one nibble. This nibble alignment continues until the L1 bit or F1 bit in an information FIFO entry is encountered. Thereafter, input to the Class 1 Alignment Block will not be nibble shifted unless the IDFNSR is written again.				

Table continues on the next page...

**Table 5-24. LOAD command DST, LENGTH, and OFFSET field values (continued)**

DST value (hex)	Class (binary)	Control data/ message data	Legal values in LENGTH /OFFSET fields	Must use IMM?	Tag	Internal register	Comment
77	00	Message	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ODFNS	Output Data FIFO Nibble Shift Register	See notes below.  Inserts the rightmost 4 bits of the immediate value into the output from a Class 1 CHA, so subsequent data from that Class 1 CHA is now shifted one nibble. Data from other sources (such as MOVE Command or LOAD IMM to the Output Data FIFO) will not be concatenated correctly. This nibble alignment continues until the CHA Done signal is asserted. Thereafter, output from a second operation, even from the same CHA, is not nibble shifted unless the ODFNSR is written again.
78	00	Message	1-8/0 bytes	Yes	AUXDATA	Auxiliary Data FIFO	See notes below.  This DST value can be used to provide data to the Auxiliary Data FIFO. Each LOAD IMM command can load 1-8 left-aligned bytes. Byte swapping is done automatically if the endianness settings require it. The LOAD IMM to AUXDATA stalls until an acknowledgment from the CCB is received. Therefore, the information FIFO entry (AST = 1 and STYPE = 00) to consume this data must ALREADY be in place before executing this command; otherwise DECO hangs. Because the command is blocking, multiple LOAD IMM commands can be used in succession, and the immediate data is consumed in order it is loaded. Note that data can also be supplied to the Auxiliary Data FIFO by using a MOVE command. MOVE commands can also be used in succession to supply data to AUXDATA, but the WC bit should be set to avoid overwriting previously supplied data.
7A	00	Control	4/0 bytes 8/0 bytes **	Yes	NFIFO	iNformation FIFO	See notes below.  The info FIFO is normally written by means of the FIFO LOAD command, but the NFIFO can be written using this DST value with a LOAD Immediate command. ** If LENGTH = 8, the LOAD command is interpreted as follows: word 1: LOAD IMMED, LENGTH = 8, DST = 7Ah word 2: bits [31:12] contain the NFIFO entry word 3: Extended Length (DECO creates as many NFIFO entries as needed to satisfy the Extended Length)
7C	00	Message	1-8/0 bytes	Yes	IFIFO	Input Data FIFO	See notes below  The input data FIFO is normally written by means of the FIFO LOAD command, but the Input Data FIFO can be written using this DST value with a LOAD Immediate Command.
7E	00	Message	1-8/0 bytes	Yes	OFIFO	Output Data FIFO	Must use a LOAD Immediate command.
All combinations of value and class that do not appear in this table are reserved							

**Table 5-25. Which Size Registers are loaded**

<b>DTYPE (hex)</b>		<b>If Class 1 CHA selected</b>	<b>If Class 2 CHA selected</b>
0		None	Class 2 Data Size
1		AAD Size and Class 1 Data Size	Class 2 Data Size
2		IV Size and Class 1 Data Size	Class 2 Data Size
3		AAD Size and Class 1 Data Size (but data only counts as Auth data)	Class 2 Data Size
4		None	Class 2 Data Size
5		None	Class 2 Data Size
6		None	Class 2 Data Size
7		None	Class 2 Data Size
8		None	Class 2 Data Size
9		None	Class 2 Data Size
A	-	ICV Size	If Both Class 1 and Class 2 CHAs selected, Class 2 Data Size else Class 2 ICV Size
C		None	Class 2 Data Size
D		None	Class 2 Data Size
E	-	None	None
F	-	Class 1 Data Size	Class 2 Data Size

## 5.6.11 FIFO LOAD command

### NOTE

In the following discussion, the term 'FIFO LOAD command' refers to both the SEQ and non-SEQ forms of the command.

FIFO LOAD commands are used to load message data, IV, AAD, ICV, and bit-length message data into the input data FIFO. The SEQ FIFO LOAD command is identical to the FIFO LOAD command except that no address is specified and the immediate bit cannot be set. See [SEQ vs non-SEQ commands](#).

As the only destination is the input data FIFO, this command does not include a DST field. Note that although this command is not blocking, it does not start until the external DMA hardware is available. The FIFO INPUT DATA TYPE is used to indicate what type of data is being loaded and whether the length is specified in bits or bytes. The length of data other than message data is measured in bytes. The length of message data can be specified in either bits or bytes (see [Table 5-29](#)). If automatic info FIFO entries are enabled, the FIFO LOAD command writes the appropriate size register(s) and the required info FIFO entry for the specified input data type.

**Table 5-26. FIFO LOAD command format**

31–27	26–25	24	23	22	21–16
CTYPE = 00100 or 00101	CLASS	SGF or VLF	IMM	EXT	INPUT DATA TYPE
15–0					
LENGTH					
31–16					
Pointer (one word, see <a href="#">Address pointers</a> ) (or Value if immediate, one or more words)					
15–0					
Pointer (or Value if immediate, one or more words)					
31–16					
EXT LENGTH (one word, present if EXT=1)					
15–0					
EXT LENGTH (one word, present if EXT=1)					

**Table 5-27. FIFO LOAD command field descriptions**

Field	Description
31-27 CTYPE	Command type 00100 FIFO LOAD 00101 SEQ FIFO LOAD
26-25 CLASS	Class. 00 For SEQ FIFO LOAD only. Skips the specified length in memory. FIFO INPUT DATA TYPE field is ignored. No info FIFO entry is generated even if automatic entries are enabled. 01 Load FIFO with data for Class 1 CHA. 10 Load FIFO with data for Class 2 CHA. 11 Load FIFO with data for both Class 1 and Class 2 CHAs (both In Snoop and Out Snoop). <b>NOTE:</b> The CLASS field must be non-zero for FIFO LOAD commands because the 0 case indicates skipping, which is illegal for FIFO LOAD. This is true even when automatic information FIFO entries are disabled.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Field (VLF) flag. Meaning depends on CTYPE. If CTYPE = 00100 (FIFO LOAD), this bit is the Scatter/Gather Table flag. 0 Pointer points to actual data. 1 Pointer points to a Scatter/Gather Table. <b>NOTE:</b> If the IMM bit is set, it is an error for this bit to be set. If CTYPE = 00101 (SEQ FIFO LOAD), this bit is the Variable Length Field flag. 0 LENGTH is length of actual data. 1 Length is variable. CAAM uses the length in the Variable Sequence In Length register and ignores the LENGTH field. <b>NOTE:</b> It is an error to set VLF = 1 when the EXT bit = 1.

*Table continues on the next page...*

**Table 5-27. FIFO LOAD command field descriptions (continued)**

Field	Description
23 IMM	Immediate Flag If CTYPE = 00100 (FIFO LOAD) 0 Data begins at the location pointed to by the next word. 1 Data follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. <b>NOTE:</b> It is an error if this bit is set when SGF = 1 and EXT = 1. If CTYPE = 00101 (SEQ FIFO LOAD) 0 Mandatory. 1 Setting this bit is an automatic error.
22 EXT	Extended Length 0 Input data length is solely determined by the 16-bit LENGTH field in the first word of the command (before the pointer) 1 Input data length is determined by the 32-bit EXTENDED LENGTH following the pointer, but the additional valid bits for bit length message data appear in the right 3 bits of the LENGTH field. See <a href="#">Bit length data</a> . <b>NOTE:</b> It is an error if this bit is set when IMM is also set.
21-16 INPUT DATA TYPE	FIFO input data type See <a href="#">Table 5-29</a> for a description of the supported types. When automatic information FIFO entries are disabled, (SEQ) FIFO LOAD Commands ignore the FIFO INPUT DATA TYPE field.
15-0 LENGTH	For EXT = 0, LENGTH = number of bytes of input data, except for bit length message data. For EXT = 1, LENGTH is ignored unless FIFO INPUT DATA TYPE = bit length. See <a href="#">Bit length data</a> .
31-0 POINTER	If IMM = 0, this field is a pointer to the data to be loaded. If IMM = 1, this field is not present. <b>NOTE:</b> This field is only present for FIFO LOAD Commands.
31-0 EXT LENGTH	For EXT = 0, this field not present. For EXT = 1, EXTENDED LENGTH specifies number of bytes to load. See <a href="#">Bit length data</a> .

### 5.6.11.1 Bit length data

If the FIFO INPUT DATA TYPE indicates that the input data type being loaded is bit length message data, the LENGTH field is defined as a bit count, as shown in the "Number of Bits" row in the following figure. This can also be interpreted as a "Number of Full Bytes field" in bits positions 15-3 , and a "Number of Additional Valid Bits" field in bit positions 2-0 . These additional valid bits are in the next byte after the number of full bytes, starting with the bit on the left. For example, if the LENGTH field is 0101h, CAAM loads 33 bytes, with only the leftmost bit of the 33rd byte valid.

**Table 5-28. Specifying data with residual bit length**

15–0	
Number of Bits (bit position 15–0)	
<b>Alternate interpretation:</b>	
15–3	2–0
Number of Full Bytes	Number of Additional Valid Bits

If the input data's bit length is equal to or greater than  $2^{16}$ , set the EXT bit and use the EXTENDED LENGTH field to specify the number of full bytes. The upper 13 bits of LENGTH must be zero, with the rightmost 3 bits specifying the number of additional valid bits as before.

### 5.6.11.2 FIFO LOAD input data type

[Table 5-29](#) contains an enumeration of the various built-in input data FIFO data types. If the CLASS field of the FIFO LOAD command does not specify a Class 1 or Class 2 destination, the corresponding Last or Flush bit is not set. Only message data can have a bit length (as opposed to a byte length), and such message data is always flushed.

Note that the information FIFO source type is not needed, as it is always the input data FIFO and the length is inferred from the amount of data being placed in the data FIFO. Also, the Last and Flush bits are always sent as 0 except with the last byte of data, in which case the values shown in the table are sent.

With the exception of IV and AAD, the FIFO LOAD command does not do any padding. This is because all algorithmic padding requires a pad length or a special last byte, which means that at least one byte of padding is required. Therefore, the padding can be sent using a padding information FIFO entry.

**Table 5-29. FIFO LOAD input data type field**

FIFO Input Type Field Bit #						Meaning	
2 1	2 0	1 9	18	17	16		
0b00	0b1111			Place the data in the input data FIFO, but do <i>not</i> generate an information FIFO entry or a data size write, even if automatic information FIFO entries are enabled.			
0b010	LC2 LC1 FC1			Message Data			
0b011	LC2 LC1 FC1			Message Data for Class 1 out-snooped to Class 2			
0b100	LC2 LC1 FC1			IV. If Last or Flush for Class 1 is set, Class 1 is padded to 16-byte boundary with 0. No padding is done for Class2			
0b101	1 LC1 FC1			Bit-length message data. Last of Class 2, Flush or Last for Class 1 must be set if the CLASS field is CLASS 1 or CLASS 1 and CLASS 2			

Table continues on the next page...

**Table 5-29. FIFO LOAD input data type field (continued)**

FIFO Input Type Field Bit #						Meaning
2 1	2 0	1 9	18	17	16	
0b110	LC2	LC1	FC1	AAD. If Last or Flush for Class 1 is set. Class 1 is padded to a 16-byte boundary with 0. No padding is done for Class 2. It is an error if Class 2 is specified and Class 1 is not.		
0b111	LC2	LC1	FC1	ICV		
LC1 means the data defined in the current information FIFO entry is the last data for the Class 1 CHA. When LC1 = 1, a flush is performed and the alignment block is emptied.						
LC2 means the data defined in the current information FIFO entry is the last data for the Class 2 CHA. When LC2 = 1, a flush is performed and the alignment block is emptied.						
FC1 means "Flush the remainder of the data out of the Class 1 alignment block."						
All values not specified are reserved.						

## 5.6.12 STORE command

STORE commands are used to read data from various registers and write them to a system address. The SEQ STORE command is identical to the STORE command except that no address is specified. In the following discussion, the term 'STORE command' refers to both the SEQ and non-SEQ forms of the command. See [SEQ vs non-SEQ commands](#).

The definitions of the OFFSET and LENGTH fields in the STORE command can depend on the CLASS and source (SRC) fields. [Table 5-30](#) shows the command fields, and [Table 5-32](#) defines OFFSET and LENGTH as well as additional behaviors of the command.

As shown in the following table, STORE data sources can be both control and message data registers. Data loaded into or stored from control data registers is regarded as word-oriented data, whereas data loaded into or stored from message data registers is regarded as byte-strings.

To facilitate operation in chips with different endianness configurations, byte-swapping and half-word swapping for control data registers can be configured independently from the swapping for message data registers. Also, the swapping can be configured independently for each job ring (see the Job Ring Configuration Register (JRCFGR)).

**Table 5-30. STORE command format**

31–27	26–25	24	23	22–16
CTYPE = 01010 or 01011	CLASS	SGF or VLF	IMM	SRC
15–8			7–10	

*Table continues on the next page...*

**Table 5-30. STORE command format (continued)**

OFFSET	LENGTH
	<b>31–16</b>
Pointer (see <a href="#">Address pointers</a> )	
	<b>15–10</b>
Pointer	
	<b>31–16</b>
If immediate (IMM = 1), one or more words of data appears here	
	<b>15–0</b>
If immediate (IMM = 1), one or more words of data appears here	

**Table 5-31. STORE command field descriptions**

Field	Description
31-27 CTYPE	Command Type 01010 STORE 01011 SEQ STORE
26-25 CLASS	Class of object to be stored The interpretation of the CLASS field depends upon the value in the SRC field. See <a href="#">Table 5-32</a> .
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Field (VLF) flag. The usage of this field depends on CTYPE. If CTYPE = 01010 (STORE), this bit is the scatter/gather table flag. 0 Pointer contains the address of the destination for the data to be stored. 1 Pointer points to a Scatter/Gather Table, which defines the destinations for the data to be stored. If CTYPE = 01011 (SEQ STORE), this bit is the Variable Length Field flag. 0 LENGTH is length of actual data. 1 Length is variable. CAAM uses the length in the Variable Sequence Out Length register and ignore the LENGTH field. <b>NOTE:</b> If the IMM bit is set, it is an error for this bit to be set.
23 IMM	Immediate data. 0 Data to be stored is found at the location defined by the SRC field. 1 Data to be stored follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. <b>NOTE:</b> It is an error if the IMM bit is set when the SGF bit is set. However, the destination of a SEQ STORE can have been defined by a Scatter/Gather Table pointed to by the SEQ OUT PTR Command that initiated the Output Sequence.) It is an error if IMM =1 and the OFFSET field is non-zero.
22-16 SRC	SRC value defines the source (CONTEXT, ICV, IV) of the data to be stored. See <a href="#">Table 5-32</a> for a list of supported sources. The SRC field is ignored if IMM = 1.
15-8 OFFSET	OFFSET defines the start point for reading within the SRC. For example, if the SRC indicates a Class 1 context, the offset can be used to indicate that the data should be read from the fourth word of context rather than from the beginning. The offset into the descriptor buffer is specified in 4-byte words, but in all other cases the offset is specified in bytes. See <a href="#">Table 5-32</a> for the legal combinations of OFFSET and LENGTH values.

*Table continues on the next page...*

**Descriptors and descriptor commands**

**Table 5-31. STORE command field descriptions (continued)**

Field	Description
7-0 LENGTH	Length of the data. For the descriptor buffer, the length is specified in 4-byte words, but in all other cases the length is specified in bytes. See <a href="#">Table 5-32</a> for the legal combinations of OFFSET and LENGTH values.
31-0 POINTER	This field is a pointer to the address in memory where the data is to be stored. <b>NOTE:</b> This field is not present for any SEQ STORE commands or for STORE commands that store the job descriptor (41h) or shared descriptor (42h) from the descriptor buffer into memory.
31-0 VALUE	If IMM = 1, the value is located here. Enough 4-byte words are used to hold the data of size LENGTH.

**Table 5-32. STORE command SRC, OFFSET and LENGTH field values**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/ OFFSET Fields	Tag	Source Internal Register	Comment
00	01	Control	4/0 bytes	MODE1	Class 1 Mode Register	-
	10			MODE2	Class 2 Mode Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DJQCR	DECO Job Queue Control Register	-
01	01	Control	4/0 bytes	KEYS1	Class 1 Key Size Register	Normally written by KEY command. Once KEY SIZE is written, the user cannot modify KEY until the key is cleared. If written by the KEY command, the Key Size register is locked.
	10			KEYS2	Class 2 Key Size Register	
	11	Control	4/0 bytes	DDAR	DECO Descriptor Address Register	-
02	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	DATAS1	Class 1 Data Size Register	Normally written by FIFO LOAD command
	10			DATAS2	Class 2 Data Size Register	
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DOPSTAT	DECO Operation Status Register	Storing DOPSTAT captures the current "math conditions" (see <a href="#">Table 5-57</a> , TEST CONDITION field, TEST CONDITION bits when JSL = 0) as well as CAAM's current idea of the command offset.

*Table continues on the next page...*

**Table 5-32. STORE command SRC, OFFSET and LENGTH field values (continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/ OFFSET Fields	Tag	Source Internal Register	Comment
03	01	Control	4/0 bytes	C1ICVS	Class 1 ICV Size Register	Normally written by FIFO LOAD command
	10			C2ICVS	Class 2 ICV Size Register	
07	00	Control	4/0 bytes	ICTRL	IRQ Control Register	-
08	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DMATH0	DECO Math Register 0	May be affected by protocols
09	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CSTA	CCB Status and Error Register	-
09	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DMATH1	DECO Math Register 1	May be affected by protocols
0A	11			DMATH2	DECO Math Register 2	
0B	01	Control	4/0 bytes	AADSZR	AAD Size Register	-
0B	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DMATH3	DECO Math Register 3	May be affected by protocols
20	01	Message	4/0-60 8/0-56 bytes	CTX1	Class 1 Context Register	A STORE from the Class 1 Context Register will automatically block until the Class 1 CHA is done.
	10			CTX2	Class 2 Context Register	A STORE from the Class 2 Context Register will automatically block until the Class 2 CHA is done.

Table continues on the next page...

**Table 5-32. STORE command SRC, OFFSET and LENGTH field values (continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/ OFFSET Fields	Tag	Source Internal Register	Comment
40	11	Control	1-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.  This SRC value can be used to store any portion of the descriptor buffer into memory. The starting word of the destination is specified in the Pointer field. The values in the LENGTH and OFFSET field are specified in 4-byte words. offset* The sum of the LENGTH and OFFSET fields must be less than or equal to 64. The OFFSET is used to specify the starting word of the source within the Descriptor Buffer. Note that the OFFSET is relative to the start of the Job Descriptor (or Trusted Descriptor) (which will not be the start of the Descriptor Buffer if there is a Shared Descriptor). See <a href="#">Figure 5-9</a> . For SEQ STORE, the data from the Descriptor Buffer will be written to the current location pointed to by the output sequence pointer (there will be no offset with respect to the destination address).
41	11	Control	1-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.  This SRC value is valid only for STORE Commands, not SEQ STORE Commands. This SRC value is used for writing back modifications to Job Descriptors (or Trusted Descriptors). This overwrites the descriptor in memory, using the address from which the descriptor was fetched. Since no Pointer is used, this is a one-word command. If an In-line descriptor, a replacement job descriptor, or a Non-local JUMP was executed, it is still the original descriptor that will be modified, not the new one. The values in the LENGTH and OFFSET field are specified in 4-byte words. offset* The sum of the LENGTH and OFFSET fields must be less than or equal to 64. The OFFSET is used to specify the starting word of the source within the Descriptor Buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the Job Descriptor (or Trusted Descriptor) (which will not be the start of the Descriptor Buffer if there is a Shared Descriptor). See <a href="#">Figure 5-9</a> .

*Table continues on the next page...*

**Table 5-32. STORE command SRC, OFFSET and LENGTH field values (continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/ OFFSET Fields	Tag	Source Internal Register	Comment
42	11	Control	1-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.
<p>This SRC value is valid only for STORE commands, not SEQ STORE commands. This SRC value is used for writing back modifications to shared descriptors. This overwrites the shared descriptor in memory, using the address from which the shared descriptor was fetched. Note that this results in an error if there is no shared descriptor. Since no pointer is used, this is a one-word command.</p> <p>The values in the LENGTH and OFFSET field are specified in 4-byte words.</p> <p>offset* The sum of the LENGTH and OFFSET fields must be less than or equal to 64. The OFFSET is used to specify the starting word of the source within the Descriptor Buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the shared descriptor in both the descriptor buffer and in memory. See <a href="#">Figure 5-9</a>.</p>						

All combinations of value and class that do not appear in [Table 5-32](#) are reserved.

### 5.6.13 FIFO STORE command

FIFO STORE commands are used solely to move data from the output data FIFO to external memory by means of the DMA. Because the only source is the output data FIFO, this command does not include a SRC field. The SEQ FIFO STORE command is identical to the FIFO STORE command except that no address is specified and the immediate bit cannot be set. See [SEQ vs non-SEQ commands](#). In the following discussion, FIFO STORE command refers to both the SEQ and non-SEQ forms of the command.

Note that data output by means of the output data FIFO is considered message data, and so is byte and half-word swapped in accordance with the message data swapping configuration. The swapping can be configured independently for each job ring (see the Job Ring Configuration Register (JRCFGR) ).

The following types of data can be output from the output data FIFO.

- S-box, which is read from the AFHA. The S-box is encrypted as a Black Key, so all operations should be completed prior to this command being executed.
- New key material that must be encrypted on its way out. This material is already in the output data FIFO, and the FIFO STORE command encrypts and stores it.

## Descriptors and descriptor commands

- Class 1 and Class 2 keys.
- RNG data, which can be left in the output data FIFO or stored away.
- Regular data, which is pulled and written as it appears in the output data FIFO. Note that bit length data stores are not available.

Note that even though this command is not blocking, it does not start if:

- Another STORE or FIFO STORE of any type is already executing
- A MOVE command is writing by means of the external DMA hardware

The FIFO LOAD command supports bit length data, (see [Bit length data](#)), but the FIFO STORE command does not support bit lengths.

It is occasionally necessary to skip over portions of the output buffer (meaning to advance the output sequence pointer without actually writing data) before writing more CHA output. For instance, in certain networking protocols, portions of the output stream may depend on out-of-order portions of the input stream. This processing can be done in two or more passes through the input and output sequences by:

1. Skipping portions of the input and output data in one pass
2. Restoring the sequences for the next pass by means of the RTO bit in the SEQ IN PTR and SEQ OUT PTR commands
3. Skipping over the portions of the output data that were written in the previous pass

To achieve skipping with SEQ FIFO STORE, use output data type 3Fh.

**Table 5-33. FIFO STORE command format**

31–27	26–25	24	23	22	21–16
CTYPE = 01100 or 01101	AUX	SGF or VLF	CONT	EXT	OUTPUT DATA TYPE
<b>15–0</b>					
LENGTH					
<b>31–16</b>					
Pointer (see <a href="#">Address pointers</a> ) (or Value if immediate)					
<b>15–0</b>					
Pointer					
<b>31–16</b>					
EXT LENGTH (EXT = 1)					
<b>15–0</b>					
EXT LENGTH (EXT = 1)					

**Table 5-34. FIFO STORE command field descriptions**

Field	Description
31-27 CTYPE	Command type 01100 FIFO STORE 01101 SEQ FIFO STORE
26-25 AUX	Auxiliary control bits. Used only for certain output data type codes. Set AUX = 00 for all other output data type codes. See <a href="#">Table 5-35</a> .
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Field (VLF) flag. Meaning depends on CTYPE. If CTYPE = 01100 (FIFO STORE), this bit is the scatter/gather table flag. 0 Pointer points to actual data. 1 Pointer points to a scatter/gather table. If CTYPE = 01101 (SEQ FIFO STORE), this bit is the Variable Length Field flag. 0 LENGTH is length of actual data. 1 Length is variable. CAAM uses the length in the Variable Sequence Out Length register and ignores the LENGTH field.
23 CONT	Continue 0 If the FIFO STORE pulls data from the output FIFO and finishes at an alignment other than at the end of a dword, the remainder of the last dword is popped and discarded. 1 This is not the last FIFO STORE command for this data. The final dword that contributed data is not popped if the data did not end at an 8-byte boundary. This is used to prevent data loss when a store leaves off in the middle of a dword. <b>NOTE:</b> If this bit is set when there is no remaining data to continue with, subsequent operations may not work as expected.
22 EXT	Use Extended Length 0 Output data length is solely determined by the 16-bit LENGTH field in the first word of the command (before the pointer). 1 Output data length is determined by the 32-bit EXT LENGTH field following the pointer.
21-16 Output Data Type	This field identifies the type of data that the output data FIFO stores. See <a href="#">Table 5-35</a> for a list of the supported types.
15-0 LENGTH	For EXT = 0, LENGTH specifies the number of bytes to store. For EXT = 1, LENGTH is ignored.
31-0 POINTER	Pointer. <b>NOTE:</b> This field is not present for SEQ FIFO STORE commands. Nor is it present for FIFO STORE commands if the data type is for RNG and the data is to be left in the output data FIFO.
31-0 EXT LENGTH	For EXT = 0, this field is not present. For EXT = 1, EXT LENGTH specifies the number of bytes to store.

[Table 5-35](#) lists the various built-in FIFO STORE output data types.

**Table 5-35. FIFO STORE output data type field**

<b>Bits 21-16 (hex)</b>	<b>Meaning</b>	<b>Comment</b>
10	AFHA S-Box, AES-CCM encrypted using the job descriptor key encryption key.	-
11	AFHA S-Box, AES-CCM encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors.
14	Key Register AES-CCM encrypted using the job descriptor key encryption key.	The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> <li>• AUX = 01 selects the Class 1 Key Register to be stored.</li> <li>• AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.</li> </ul>
15	Key register, AES-CCM encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> <li>• AUX = 01 selects the Class 1 Key Register to be stored.</li> <li>• AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.</li> </ul>
16	Class 2 Key Register MDHA Split Key. AES-CCM encrypted using the job descriptor key encryption key.	For performance and security, use of an MDHA split key is highly recommended. Details about split keys can be found in <a href="#">Using the MDHA Key Register with IPAD/OPAD "split keys"</a> . The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see <a href="#">MDHA use of the Context Register</a> ).
17	Class 2 Key Register MDHA Split Key, AES-CCM encrypted using the trusted descriptor key encryption key. Only available to trusted descriptors.	
18	Output Data FIFO AES-CCM encrypted using the job descriptor key encryption key	Limited to material that fits in the output data FIFO. Can be used only by the built-in protocols.
19	Output Data FIFO, AES-CCM encrypted using the trusted descriptor key encryption key	Limited to material that fits in the Output Data FIFO. Can be used only by the built-in protocols. Only available to trusted descriptors.
20	AFHA S-Box, AES-ECB encrypted using the job descriptor key encryption key	-
21	AFHA S-Box, AES-ECB encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors.
24	Key Register, AES-ECB encrypted using the job descriptor key encryption key.	The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> <li>• AUX = 01 selects the Class 1 key register to be stored.</li> <li>• AUX = 10 selects the Class 2 key register to be stored. AUX values 00 and 11 are illegal.</li> </ul>
25	Key Register, AES-ECB encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> <li>• AUX=01 selects the Class 1 Key Register to be stored.</li> <li>• AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.</li> </ul>

*Table continues on the next page...*

**Table 5-35. FIFO STORE output data type field (continued)**

Bits 21-16 (hex)	Meaning	Comment
26	Class 2 Key Register MDHA split key, AES-ECB encrypted using the job descriptor key encryption key.	For performance and security, use of an MDHA split key is highly recommended. Details on this split key can be found in <a href="#">Using the MDHA Key Register with IPAD/OPAD "split keys"</a> . The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see <a href="#">MDHA use of the Context Register</a> ).
27	Class 2 Key Register MDHA split key, AES-ECB encrypted using the trusted descriptor key encryption key. Only available to trusted descriptors.	
28	Output Data FIFO, AES-ECB encrypted using the job descriptor key encryption key	Limited to material that fits in the output data FIFO. Can be used only by the built-in protocols.
29	Output Data FIFO, AES-ECB encrypted using the trusted descriptor key encryption key	Limited to material that fits in the output data FIFO. Can be used only by the built-in protocols. Available only to trusted descriptors.
30	Message Data	-
34	Store the specified amount of data to be obtained from RNG to the system address provided. Extended lengths are illegal.	<b>NOTE:</b> The Class 1 Data Size Register is automatically written.  The different types of random data that can be generated are: <ul style="list-style-type: none"> <li>• Random data with no restriction</li> <li>• Nonzero Random data</li> <li>• Odd Parity Random data.</li> </ul> The Mode Register controls the type of random data. Note that the RNG must be selected by writing the Mode register.
35	Obtain the specified amount of data from RNG and leave it in the output data FIFO. Extended lengths are illegal and there is no pointer.	
3F	Skip	Skip over the specified length in memory without using bus cycles. Permitted to be used only by SEQ FIFO STORE.
<b>NOTE:</b> AUX must be set to 00 except when otherwise specified above. All combinations of output data type and AUX values not specified are reserved.		

## 5.6.14 MOVE and MOVE\_LEN commands

The MOVE command is used to copy data between two resources internal to CAAM. This allows data to be put in the proper registers without having to store data to external memory and then load it back in again.

The OFFSET field is used to define an offset into either the source or destination, depending on the values in the SRC, DST, and AUX fields (see table in the OFFSET field description below). The MOVE command has a limited number of sources and destinations as indicated in the SRC and DST field descriptions below.

MOVE cautions and restrictions:

- Keys can't be copied from a key register by means of a MOVE command.

## Descriptors and descriptor commands

- Observe the cautions noted in the "RJD" field of the SEQ IN PTR command ([Table 5-63](#)) if using a MOVE command in a Replacement Job Descriptor.
- Moves to the Input Data FIFO may be checkpoints. For example, a move from the Class 2 Context Register to the Input Data FIFO for the Class 1 CHA is a Load Checkpoint and is a Done Checkpoint for the Class 2 CHA.

In the MOVE command the 8-bit LENGTH field specifies the amount of data to be moved (see the table in the LENGTH field description below). The MOVE\_LEN command is identical to the MOVE command except that the length of the data being moved is specified by a MATH register, rather than specified as a constant within the command. In the MOVE\_LEN command the MRSEL (Math Register Select) field replaces the MOVE command's LENGTH field. In this section "MOVE Command" is used to refer to both the MOVE and MOVE\_LEN forms of the command.

Note that the AUX field is used to select among a number of different options, depending on the values in the SRC, DST and AUX fields (see the table in the AUX field description below).

**Table 5-36. MOVE command format**

		31–27	26–25	24	23–20				19–16				
		CTYPE = 01111 or 01110	AUX	WC	SRC				DST				
		15–8				7	6	5	4	3	2	1–0	
MOVE format		OFFSET				LENGTH							
MOVE_LEN format		OFFSET				0	0	0	0	0	0	0	MRSEL

**Table 5-37. MOVE command field descriptions**

Field	Description
31-27 CTYPE	Command Type 01111 MOVE. Performs an internal move between two internal CAAM locations. The length of the data is specified by the value in the LENGTH field. 01110 MOVE_LEN. Performs an internal move between two internal CAAM locations. The length of the data is specified by the value in the MATH register selected by the MRSEL field.
26-25 AUX	AUX bits are used for some SRC and DST combinations to specify additional options. See <a href="#">Table 5-38</a> below.
24 WC	Wait for Completion 0 Do not Wait for Completion 1 Wait for Completion. Causes the MOVE to stall until the MOVE completes. (A separate command to wait is not needed in those cases where a wait is needed.)

*Table continues on the next page...*

**Table 5-37. MOVE command field descriptions (continued)**

Field	Description
23-20 SRC	Source. This specifies the internal source of data that will be moved. Not all combinations of source and destination are allowed. See <a href="#">Table 5-40</a> below. <a href="#">Table 5-38</a> and <a href="#">Table 5-39</a> indicate which source and destination combinations are permitted.
19-16 DST	Destination This specifies the internal destination of the data that will be moved. The tables in the AUX and OFFSET field descriptions indicate which source and destination combinations are permitted.
15-8 OFFSET	Offset. (in bytes) The interpretation of the OFFSET field depends on the source and destination, as shown below. The OFFSET is limited to 128 bytes except when the Descriptor Buffer is the source or destination, in which case the OFFSET may be as large as 255 bytes.
7-0 LENGTH	Length for internal move. (in bytes, 128 max).
7-2	Reserved These bits are reserved in the MOVE_LEN form of the command. These bits and the MRSEL field below replace the LENGTH field that appears in the MOVE form of the command.
1-0 MRSEL	MATH Register Select This field is used only in the MOVE_LEN form of the command. The MRSEL field and the reserved bits above replace the LENGTH field that appears in the MOVE form of the command. In the MOVE_LEN form of the command the length (in bytes) of the data to be moved is specified in the MATH Register selected by the MRSEL field. If the move is from the input FIFO to the output FIFO, bits 15:0 of the MATH Register are used for the length; otherwise, only bits 7:0 are used. Other bits are simply ignored. 00 Math Register 0 01 Math Register 1 10 Math Register 2 11 Math Register 3

**Table 5-38. Usage of the AUX field in the MOVE and MOVE\_LEN Commands**

DST → SRC ↓	0h: C1 Context 1h: C2 Context	2h: Output Data FIFO	3h: Descriptor Buffer	4h: Math 0 5h: Math 1 6h: Math 2 7h: Math 3	8h: Class 1 Input Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: Reserved	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
-------------------	----------------------------------	----------------------	-----------------------	--	-----------------------------	-----------------------------	--	--------------	--------------------------	-------------------

*Table continues on the next page...*

**Descriptors and descriptor commands**

**Table 5-38. Usage of the AUX field in the MOVE and MOVE\_LEN Commands (continued)**

0h: C1 Context 1h: C2 Context			AUX selects offset into Context Register 00: 0 bytes 01: 16 bytes 10: 32 bytes 11: 48 bytes		AUX <sub>MS</sub> : Flush AUX <sub>LS</sub> : Last	AUX <sub>LS</sub> : Last			AUX <sub>LS</sub> =0 : OFFSET field into Context Reg AUX <sub>LS</sub> =1 : OFFSET into Key Reg	
2h: Output FIFO		move not allowed								
3h: Descr Buffer	AUX selects offset into Context Register 00: 0 bytes 01: 16 bytes 10: 32 bytes 11: 48 bytes		move not allowed							
4h: Math Reg 0 5h: Math Reg 1 6h: Math Reg 2 7h: Math Reg 3										

*Table continues on the next page...*

**Table 5-38. Usage of the AUX field in the MOVE and MOVE\_LEN Commands (continued)**

8h: DECO Alignment Block (flushed)					move not allowed					move not allowed
9h: Class 1 or Class 2 Alignment Block	AUX <sub>MS</sub> : Flush AUX <sub>LS</sub> =0 : Source is Class 2 Alignment Block AUX <sub>LS</sub> =1 : Source is Class 1 Alignment Block								AUX <sub>MS</sub> : Flush AUX <sub>LS</sub> =0 : Source is C2 Align Block AUX <sub>LS</sub> =1 : Source is C1 Align Block	
Ah: DECO, Class 1 or Class 2 Alignment Block	AUX field selects Alignment Block 00: DECO Alignment Block 01: Class 1 Alignment Block 10: Class 2 Alignment Block 11: error								AUX field selects Align Block 00: DECO Align Block 01: C1 AB 10: C2 AB 11: error	

**Table 5-39. Usage of the OFFSET field in the MOVE and MOVE\_LEN Commands**

DST → SRC ↓	0h: C1 Context 1h: C2 Context	2h: Output Data FIFO	3h: Descriptor Buffer	4h: Math 0 5h: Math 1 6h: Math 2 7h: Math 3	8h: Class 1 Input Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: Reserved	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
0h: C1 Context 1h: C2 Context	OFFSET field is used for SRC	OFFSET field is used for SRC (must be a multiple of 4)	OFFSET field is used for DST (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC				AUX <sub>LS</sub> =0: OFFSET field is used for Context Reg AUX <sub>LS</sub> =1: OFFSET field is used for Key Reg	OFFSET field is used for SRC (must be a multiple of 4)	
2h: Output FIFO	OFFSET field is used for DST	move not allowed	OFFSET field is used for DST		Error generated if OFFSET≠0				OFFSET field is used for DST	unused

Table continues on the next page...

**Descriptors and descriptor commands**

**Table 5-39. Usage of the OFFSET field in the MOVE and MOVE\_LEN Commands (continued)**

3h: Descr Buffer	OFFSET field is used for SRC (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC (must be a multiple of 4)	move not allowed	OFFSET field is used for SRC	OFFSET field is used for SRC	OFFSET field is used for SRC (must be a multiple of 4)
4h: Math Reg 0 5h: Math Reg 1 6h: Math Reg 2 7h: Math Reg 3	OFFSET field is used for DST	OFFSET field is used for SRC (must be a multiple of 4)	OFFSET field is used for DST	OFFSET field is used for SRC	OFFSET field is used for DST	OFFSET field is used for SRC (must be a multiple of 4)
8h: DECO Align Block (flushed) 9h: Class 1 or Class 2 Alignment Block Ah: DECO, Class 1 or Class 2 Alignment Block	OFFSET field is used for DST (must be a multiple of 4)	OFFSET is ignored in MOVE_LEN In MOVE, OFFSET field is prepended to the LENGTH field to form a 16-bit length	OFFSET field is used for DST (must be a multiple of 4)	move not allowed	OFFSET field is used for DST (must be a multiple of 4)	move not allowed

**Table 5-40. Move source descriptions**

Value	Move Source	Notes
0h	Class 1 Context Reg	—
1h	Class 2 Context Reg	—
2h	Output Data FIFO	—
3h	Descriptor Buffer	—
4h	Math Register 0	—
5h	Math Register 1	—
6h	Math Register 2	—
7h	Math Register 3	—
8h	DECO Alignment Block (always Flushed)	forces CCB DMA; Input to the DECO Alignment Block is specified by an iNfo FIFO entry.

*Table continues on the next page...*

**Table 5-40. Move source descriptions (continued)**

Value	Move Source	Notes
9h	Class 1 or Class 2 Alignment Block	<p>forces CCB DMA; The choice between the Class 1 and Class 2 Alignment Blocks is determined by the least-significant bit of the AUX field:</p> <ul style="list-style-type: none"> <li>• <math>A_{LS} = 0</math> selects C2 Alignment Block</li> <li>• <math>A_{LS} = 1</math> selects C1 Alignment Block</li> </ul> <p>Input to the Class 1 or Class 2 Alignment Block is specified by an iNfo FIFO entry. The most-significant bit of the AUX field must be set (causing a FLUSH) only if the destination is the Output Data FIFO.</p>
Ah	DECO, Class 1 or Class 2 Alignment Block, as specified via the AUX field.	<p>forces CCB DMA; no iNformation FIFO entry generated;</p> <p>AUX = 00b: use DECO Alignment Block</p> <p>AUX = 01b: use Class 1 Alignment Block</p> <p>AUX = 10b: use Class 2 Alignment Block</p> <p>AUX = 11b: error</p> <p>Input to the Class 1 or Class 2 Align Block is specified by an iNfo FIFO entry.</p>
All other values are reserved		

**Table 5-41. Move destinations**

Value	Move Destination	Notes
0h	Class 1 Context	—
1h	Class 2 Context	—
2h	Output Data FIFO	Forces CCB DMA
3h	Descriptor Buffer	—
4h	Math Register 0	—
5h	Math Register 1	—
6h	Math Register 2	—
7h	Math Register 3	—
8h	Input Data FIFO (C1)	If Automatic iNformation FIFO entries are enabled, the entries are generated for a Class 1 CHA.
9h	Input Data FIFO (C2)	If Automatic iNformation FIFO entries are enabled, the entries are generated for a Class 2 CHA.
Ah	Input Data FIFO	No iNformation FIFO entry generated
Dh	Class 1 Key	—
Eh	Class 2 Key	—
Fh	Auxiliary Data FIFO	Data can be moved to the Auxiliary Data FIFO so that it can later be used as an input to one or more of the Alignment Blocks. (An NFIFO entry with AST = 1 and STYPE = 00 must be created before the MOVE, else DECO may hang.) Note that a LOAD IMM to destination 78 can also be used to supply data to the Auxiliary Data FIFO. If multiple MOVEs and/or MOVEs and LOADs are used to provide data to the Auxiliary Data FIFO, the MOVE commands may need the WC bit set to ensure that the data is not overwritten.
All other values are reserved		

## 5.6.15 ALGORITHM OPERATION command

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that CAAM performs. Setting OPTYPE = 010 or 100 specifies an ALGORITHM OPERATION. Setting OPTYPE = 000, 110, or 111 specifies a PROTOCOL OPERATION (see [PROTOCOL OPERATION commands](#)). The operation can range from performing a simple operation using a single CHA to performing a complex operation involving multiple CHAs and multiple steps. More than one OPERATION command can be used in a descriptor, allowing Class 1 and Class 2 operations to be specified separately. For the ALGORITHM OPERATION command, the fields of the command are as shown in the following table. Note that bits 23-0 of the ALGORITHM OPERATION command are automatically written to the appropriate CHA's mode register.

**Table 5-42. ALGORITHM OPERATION command format**

—	31–27								26–24				23–16			
	CTYPE = 10000								OPTYPE = 010 or 100				ALG			
	15	14	13	12	11	10	9	8	7	6	5	4	3–2	1	0	
format for CHAs other than RNG	0	0	0	AAI								AS	ICV	ENC		
format for RNG	0	0	0	SK	AI	PS	OBP	NZB	0	0	SH	AS	PR	TST		

**Table 5-43. ALGORITHM OPERATION command field descriptions**

Field	Description
31-27 CTYPE	Command type 10000 OPERATION
26-24 OPTYPE	Operation Type If OPTYPE = 2h (Class 1 algorithm) or OPTYPE = 4h (Class 2 algorithm), the ALG, AAI, AS, ICV, and ENC field are interpreted as shown in the appropriate field description below.

*Table continues on the next page...*

**Table 5-43. ALGORITHM OPERATION command field descriptions (continued)**

Field	Description
23-16 ALG	<p>Algorithm</p> <p>This field specifies the algorithm that is to be used for the operations.</p> <ul style="list-style-type: none"> <li>• If OPTYPE = 2h           <ul style="list-style-type: none"> <li>• 10h AES</li> <li>• 20h DES</li> <li>• 21h 3DES</li> <li>• 30h ARC4</li> <li>• 50h RNG</li> <li>• All other values are reserved.</li> </ul> </li> <li>• If OPTYPE = 4h           <ul style="list-style-type: none"> <li>• 40h MD5</li> <li>• 41h SHA-1</li> <li>• 42h SHA-224</li> <li>• 43h SHA-256</li> <li>• 44h SHA-384</li> <li>• 45h SHA-512</li> <li>• All other values are reserved.</li> </ul> </li> </ul>
12-4 AAI	<p>Additional Algorithm Information</p> <p>This field contains additional mode information that is associated with the algorithm that is being executed. See <a href="#">Table 5-44</a> below for details. See also the section describing the appropriate CHA. Note that some algorithms do not require additional algorithm information and in those cases this field should be all 0s.</p>
3-2 AS	<p>Algorithm State</p> <p>This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see the shaded AS field below.</p> <p>00 Update 01 Initialize 10 Finalize 11 Initialize/finalize</p>
1 ICV	<p>ICV Checking</p> <p>For the definition of this bit in RNG commands, see the shaded PR field below. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit is ignored by algorithms that do not support ICV checking.</p>
0 ENC	<p>Encrypt/Decrypt</p> <p>For the definition of this bit in RNG commands, see the shaded TST field below. This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes.</p> <p>0 Decrypt 1 Encrypt</p>
The shaded rows below describe how bits 12-0 are interpreted for RNG commands.	
12 SK	<p>Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS≠00.</p>

*Table continues on the next page...*

**Descriptors and descriptor commands**

**Table 5-43. ALGORITHM OPERATION command field descriptions (continued)**

Field	Description
11 AI	Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the 256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Unstantiate).
10 PS	Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional "entropy" input during instantiation. Note that the personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01.
9 OBP	Odd Byte Parity. For RNG OPERATION commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00.
8 NZB	NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00.
7-6	Reserved. For RNG commands these bits of the AAI field are reserved.
5-4 SH	State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented.  00 State Handle 0 01 State Handle 1 10 Reserved 11 Reserved
3-2 AS	Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown in <a href="#">Table 5-45</a> :
1 PR	Prediction Resistance. For RNG OPERATION commands this bit is interpreted as shown in <a href="#">Table 5-46</a> :
0 TST	Test Mode Request. For RNG OPERATION commands this bit is interpreted as shown in <a href="#">Table 5-47</a> .

Note that for RNG OPERATION commands the AAI field is interpreted as shown in the shaded SK, AI, PS, OBP, NZ and SH fields above.

**Table 5-44. AAI interpretation for AES and DES modes**

AAI Interpretation for AES Modes ( <a href="#">AES accelerator (AESA) functionality</a> )				AAI Interpretation for DES ( <a href="#">Data encryption standard accelerator (DES) functionality</a> )									
<b>[For AES the MSB of AAI is the DK (Decrypt Key) bit.]</b>													
Code	Interpretation	Code 1	Interpretation	Code	Interpretation	Value	Interpretation						
00h	CTR (mod $2^{128}$ )	80h	CCM (mod $2^{128}$ )	10h	CBC	30h	CFB						
10h	CBC	90h	Reserved	20h	ECB	40h	OFB						
20h	ECB	A0h	CBC_XCBC_MAC	80h ORed with any DES code above: Check odd parity									
30h	CFB	B0h	CTR_XCBC_MAC										
40h	OFB	C0h	CBC_CMAC	<b>AAI Interpretation for RNG (<a href="#">Random-number generator (RNG) functionality</a>)</b>									
50h	Reserved	D0h	CTR_CMAC_LTE	For RNG operations, see shaded rows in the table above for the definitions of the 13 ls bits of the Class 1 Mode Register.									
60h	CMAC	E0h	CTR_CMAC										
70h	XCBC-MAC												
Setting the DK bit (i.e. ORing 100h with any AES code above) causes Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in <a href="#">AES accelerator (AESA) functionality</a> .													
<b>AAI Interpretation for MD5, SHA-1, SHA-256, SHA-384 &amp; SHA-512 (<a href="#">Message digest hardware accelerator (MDHA) functionality</a>)</b>													
Code <sup>1</sup>	Interpretation	Value	Interpretation										
00h	Hash without key	04h	HMAC with precomputed IPAD and OPAD										
01h	HMAC												

- The codes in the column are mutually exclusive, which means that they cannot be ORed with each other.

**Table 5-45. AS RNG OPERATION command settings**

AS Value	State Handle is already instantiated	State Handle is NOT already instantiated
00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error
01 Instantiate	Error	Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.
10 Reseed	Reseed the state handle.	Error
11 Unstantiate	Unstantiate the state handle.	Error

**Table 5-46. PR RNG Operation commands setting**

AS Value	PR = 0	PR = 1
00 Generate	Do NOT reseed prior to generating new random data	If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.
01 Instantiate	Instantiate the state handle to NOT support prediction resistance	Instantiate the state handle to support prediction resistance
10 Reseed	Reseed the state handle. PR bit is ignored.	Reseed the state handle. PR bit is ignored.
11 Uninstantiate	Uninstantiate the state handle. PR bit is ignored.	Uninstantiate the state handle. PR bit is ignored.

**Table 5-47. TST RNG OPERATION command settings**

AS Value	TST = 0	TST = 1
00 Generate	<ul style="list-style-type: none"> <li>If the selected state handle is in nondeterministic mode, generate new random data.</li> <li>If the selected state handle is in deterministic mode, generate a Test error.</li> </ul>	<ul style="list-style-type: none"> <li>If the selected state handle is in deterministic mode, generate new random data.</li> <li>If the selected state handle is in nondeterministic mode, generate a Test error.</li> </ul>
01 Instantiate	Instantiate the state handle in normal (nondeterministic) mode.	Instantiate the state handle in test (deterministic) mode.
10 Reseed	<ul style="list-style-type: none"> <li>If the selected state handle is in nondeterministic mode, reseed the state handle.</li> <li>If the selected state handle is in deterministic mode, generate a Test error.</li> </ul>	<ul style="list-style-type: none"> <li>If the selected state handle is in nondeterministic mode, reseed the state handle.</li> <li>If the selected state handle is in deterministic mode, generate a Test error.</li> </ul>
11 Uninstantiate	Uninstantiate the state handle. TST bit is ignored.	Uninstantiate the state handle. TST bit is ignored.

## 5.6.16 PROTOCOL OPERATION commands

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that CAAM performs. The OPERATION command's Protocol OpType takes advantage of well-known processing steps for standardized security protocols, so that the user can invoke an existing hard coded command sequence rather than having to use SEQ commands to create a complex descriptor.

If the OPTYPE is a protocol (000, 110, 111), the OPERATION command fields are as shown in [Table 5-48](#). If OPTYPE specifies an algorithm operation (OPTYPE = 010: Class 1, OPTYPE = 100: Class 2), see [ALGORITHM OPERATION command](#).

**Table 5-48. PROTOCOL OPERATION command format**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

*Table continues on the next page...*

**Table 5-48. PROTOCOL OPERATION command format (continued)**

CTYPE = 10000					OPTYPE = 000, 110, or 111				PROTID										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PROTINFO			

Protocols are used to execute complex built-in functions. Before beginning a protocol operation, DECO waits for any pending (SEQ) FIFO STORE operations to complete. When starting the protocol operation, DECO resets the output data FIFO; any data remaining in the output data FIFO from previous operations is lost. It is the responsibility of the programmer to ensure that once the protocol starts, no data is pushed into the output data FIFO as a result of commands executed prior to the protocol operation. It is the responsibility of the programmer to ensure that once the protocol starts, no data is in, or will be pushed into, the input data FIFO or information FIFO as a result of commands executed prior to the protocol operation.

The protocol information codes are shown in the PROTINFO field in [Table 5-50](#).

**Table 5-49. PROTOCOL OPERATION command field descriptions**

Field	Description
31-27 CTYPE	Command type  The total list of command types can be found in <a href="#">Table 5-13</a> . To perform an OPERATION command, set CTYPE to 0b10000.
26-24 OPTYPE	OPTYPE  000 Unidirectional protocol  110 Decapsulation protocol  111 Encapsulation protocol  All others: reserved or not protocols
23-16 PROTID	Protocol Identifier  This field represents the specific protocol that this descriptor is implementing.

**Table 5-50. PROTINFO values**

Field	Description
PROTINFO 15-0	This value is protocol-dependent.  • For blobs encapsulation or decapsulation, the PROTINFO field is defined in <a href="#">Table 5-51</a> and <a href="#">Table 5-52</a> . For further information concerning blobs, see <a href="#">Blobs</a> .

**Table 5-51. Blob PROTINFO field**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1:0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	-----

*Table continues on the next page...*

**Table 5-51. Blob PROTINFO field (continued)**

0	0	0	0	0	0	TK	EKT	K2KR	Sec_Mem	Black_Key	Blob_Format
---	---	---	---	---	---	----	-----	------	---------	-----------	-------------

**Table 5-52. Blob PROTINFO field descriptions**

Field	Description
15-10	Reserved.
9 TK	Trusted Key Used only for trusted descriptors with black blob encapsulation/decapsulation. Ignored otherwise. 0 Use the JDKEK when encrypting or decrypting black keys. 1 Use the TDKEK when encrypting or decrypting black keys.
8 EKT	Encrypted Key Type Used only for black blob encapsulation/decapsulation. Ignored otherwise. 0 Use AES-ECB mode when encrypting/decrypting black keys. 1 Use AES-CCM mode when encrypting/decrypting black keys.
7-4 K2KR	Key to Key Register Specifies the destination for the result of black blob decapsulation. Ignored otherwise. Black blob encapsulation always uses a source from memory. The source and destination for red blob encapsulation and decapsulation is always memory. (See <a href="#">Blob types differentiated by content</a> ) 0000 Memory 0001 Class 1 key register 0011 Class 2 key register 0111 Class 2 key register (split key) All other values are reserved.
3 Sec_Mem	Location of plaintext data 1 Plaintext data is in a Secure Memory partition. 0 Plaintext data is in general purpose memory external to CAAM.
2 Black_key	0 Red Blob. The data encapsulated into the blob or decapsulated from the blob is treated as plaintext. 1 Black Blob. The data encapsulated into the blob or decapsulated from the blob is treated as a black key encrypted with the appropriate KEK (JDKEK or TDKEK). For blob encapsulation operations, the input data is first decrypted using the appropriate KEK and then encrypted using the blob key. For blob decapsulation operations, the data portion of the blob is decrypted using the blob key. If the resulting plaintext is to be written into memory rather than into a key register, the plaintext is encrypted using the appropriate KEK.

*Table continues on the next page...*

**Table 5-52. Blob PROTINFO field descriptions (continued)**

Field	Description
1-0 Blob_Format	<p>The format of the blob.</p> <p>00 Normal Blob. The output is composed of the encrypted blob key, the encrypted data, and MAC tag.</p> <p>01 Reserved</p> <p>10 Master Key Verification Blob. The master key is used for key derivation, regardless of security mode. Only the derived blob key encryption key is output. Note that the Blob_Format value is an input to the BKEK derivation, which ensures that the BKEK value that is exposed in a master key verification blob is different than the BKEK value used for any other blob format. Furthermore, the use of SHA-256 in BKEK derivation ensures that the BKEK values used for other blob formats cannot be learned by analyzing the BKEK values used for master key verification blobs.</p> <p>11 Test Blob. The non-volatile test key is used for key derivation. The output is composed of the derived blob key encryption key, the actual blob key, the encrypted blob key, the encrypted data, and MAC tag. Test blobs can be exported or imported only when CAAM is in non-secure mode.</p>

## 5.6.17 SIGNATURE command

Trusted descriptors end with a SIGNATURE command, which requires the descriptor's signature (HMAC) to be validated before allowing it to run. SIGNATURE commands also support regeneration of the signature if the trusted descriptor modifies itself.

Trusted descriptors can be created and signed with a signature (a keyed hash) when executed from a specially privileged job ring. (See [Trusted descriptors](#).) Trusted descriptors can be used to integrity protect the descriptor and to bind a key to a descriptor.

The SIGNATURE command that generates and verifies the signed hash is always the last command of a trusted descriptor, although additional SIGNATURE commands can appear within the descriptor. The signature (HMAC) immediately follows the last SIGNATURE command in the trusted descriptor. When the descriptor is created:

- Room must be left at the end of the buffer for the 32-byte signature
- The length of the descriptor must include the signature.

DECO does not read the signature when creating the signature, so any initial value can be placed there.

If a trusted descriptor has a shared descriptor, the shared descriptor is part of the signed hash computation. The shared descriptor is hashed first, followed by the descriptor; this is the order in which they appear in the descriptor buffer. The final hash is the value computed for both.

**NOTE**

It is an error for a SIGNATURE command to be in a descriptor that is not trusted or being made trusted.

**NOTE**

Because the SIGNATURE command must be the last command executed in the descriptor, trusted descriptors cannot have the REO bit set in their header. Doing so results in an error.

SIGNATURE types are available that allow a portion of the following command to not be included in the signed hash. This provides flexibility in changing the address or the immediate data specified by a command. The writer of the trusted descriptor is responsible for using these SIGNATURE types only when the skipped information does not need to be integrity protected, meaning any immediate data or any address is permissible.

**NOTE**

Skipping the signature over immediate data can allow a malicious user to shorten the length of the immediate data and insert additional commands that would not be included in the signed hash. Note that this could be done without altering the overall length of the descriptor. To prevent this, it is recommended that the first four bytes of an immediate command always be protected by the signed hash. Because the length of the immediate data is included in the signed hash, the length cannot be altered such that additional commands can substitute for a portion of the immediate data.

**Table 5-53. SIGNATURE command, format**

31–27					26	25	24	23	22	21	20	19–16			
CTYPE = 10010					0	0	0	0	0	0	0	TYPE			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31–16															
8 Words to hold the Signature (types 0000, 0001 and 0010 only)															
15–0															
8 Words to hold the Signature (types 0000, 0001 and 0010 only)															

**Table 5-54. SIGNATURE command, field descriptions**

Field	Description
31-27	Command type
CTYPE	10010 Signature
26-20	Reserved
19-16	See <a href="#">Table 5-55</a>
TYPE	
15-0	Reserved

**Table 5-55. TYPE bit settings**

Type	Meaning	Instructions
0000	SIGNATURE command types 0000, 0001, or 0010 must be the last command that is executed in a trusted descriptor. If one of these types is used, the trusted descriptor signature (the keyed hash value) immediately follows the command. It is an error for a SIGNATURE command with one of these types to appear anywhere other than at the end of the descriptor.	-
0001		Type 0001 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution. This type is used in cases where the descriptor could modify itself during execution
		Type 0010 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution if, upon completion, the MATH_Z bit is set. This type is used in cases where the descriptor could modify itself during execution but updating the keyed hash should be conditional.
1010	SIGNATURE command types 1010, 1011, and 1100 are used to include only a portion of the following instruction in the keyed hash calculation, omitting the remainder of the instruction from the calculation. There is no hash value associated with this type, so it is an error for this type to appear at the end of the descriptor. These types allow the trusted descriptor to be modified with other offsets, addresses and lengths without invalidating the signature. Note that the SIGNATURE command is, itself, included in the hash so that it cannot be added later.	Type 1010 instructs CAAM to hash only the first 2 bytes of the immediately following command.
1011		Type 1011 instructs CAAM to hash only the first 3 bytes of the immediately following command.
1100		Type 1100 instructs CAAM to hash only the first 4 bytes of the immediately following command.
Others	Reserved	

## 5.6.18 JUMP (HALT) command

The JUMP command has the following uses:

- Alters the execution order of descriptor commands
- Halts the execution of the descriptor if specified conditions occur.

[Table 5-56](#) shows the format of the JUMP command, and [Table 5-57](#) describes the JUMP command field definitions.

### 5.6.18.1 Jump type

The JUMP command has six different types, distinguished by the value in the JUMP TYPE field. All of these types specify a tested condition by means of the TEST CONDITION field. See [Test type](#) for an explanation of what it means for the tested condition to be true.

Two of these jump types are true conditional jumps, another two are actually conditional halts, and the last two are a conditional subroutine call and a conditional subroutine return. Regardless of the jump type, the execution of the command waits for any specified wait conditions to be satisfied before the conditional action (jump, call, return, or halt) is taken. Some wait conditions can be specified in the CLASS field (wait for the Class 1 CHA to be done, wait for the Class 2 CHA to be done, or both), and additional wait conditions can be specified with the TEST CONDITION field (if JSL = 1).

#### 5.6.18.1.1 Local conditional jump

The local conditional jump works as follows:

- If the tested condition is true, a JUMP command of the local conditional jump type continues the execution sequence at a new point within the descriptor buffer.
- If the tested condition is false, the jump is not taken and execution instead continues with the command that follows the JUMP command.

Local jumps are relative. The LOCAL OFFSET field is treated as an 8-bit 2's complement number that is added to the position of the JUMP command within the descriptor buffer. That is, a jump of one goes to the next 32-bit word, a jump of two skips one 32-bit word, etc. Backward jumps are performed using 2's complement negative numbers.

A LOCAL OFFSET of 0 is a shorthand means of jumping back to the start of the descriptor buffer, which is either the start of the job descriptor if there is no shared descriptor or the start of the shared descriptor, if there is one. (see [Figure 5-9](#))

#### 5.6.18.1.2 Non-local conditional jump

The non-local conditional jump is just like the local conditional jump, except that the target of the jump must be the header of a job descriptor or trusted descriptor (not a shared descriptor). The pointer to the target job descriptor is in the word following the JUMP command.

- If the tested condition evaluates to true, the jump is taken.
- If the tested condition evaluates to false, the jump is not taken and execution continues with the command following the pointer.

The target of a non-local JUMP is not permitted to contain a shared descriptor.

#### **NOTE**

It is permissible to JUMP from a job descriptor to another job descriptor or from a job descriptor or a trusted descriptor to another trusted descriptor, but jumping from a trusted descriptor to a job descriptor results in an error.

The JUMP command may or may not be a checkpoint depending on its conditions and type.

#### **5.6.18.1.3 Conditional halt**

This JUMP command is actually a conditional halt, meaning it stops the execution of the current descriptor if the tested condition evaluates to true. In this case the Math condition bits (see the "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field in [Table 5-56](#)) are written out right-justified in the SSED field of the job termination status word (see [Job termination status/error codes](#)).

If the tested condition evaluates to false, the descriptor is not halted and execution instead continues with the command that follows the jump.

#### **5.6.18.1.4 Conditional halt with user-specified status**

A JUMP command with the user-specified status option is another type of conditional halt. If the tested condition is true, it stops execution of the descriptor but instead of writing the Math condition bits, this conditional halt writes out the value in the LOCAL OFFSET field (again, right-justified in the SSED field of the job termination status word). The interpretation of the code in the LOCAL OFFSET field is user-specified, so it could be used during debugging to indicate that execution reached a certain point in a particular descriptor. If the tested condition evaluates to false, execution continues with the command following the jump.

### 5.6.18.1.5 Conditional subroutine call

A JUMP command with the subroutine call option is another type of local conditional jump. If the tested condition is true, it jumps to the specified location in the descriptor buffer but also saves the return address in the Return Address register. The return address is the location immediately following the JUMP command. If the tested condition evaluates to false, execution continues with the command following the JUMP.

Note that there is only one Return Address register, so subroutine calls cannot be nested. Because the built-in protocols make extensive use of the subroutine call type of jump, a subroutine cannot include a protocol operation.

### 5.6.18.1.6 Conditional subroutine return

A JUMP command with the subroutine return option is another type of local conditional jump. In this case the local offset is ignored because the target of the jump is taken from the Return Address register. If the tested condition is true, the subroutine return jumps to the address specified in the Return Address register. This address is the location immediately following the most recent subroutine call JUMP in which the tested condition evaluated as true. If the tested condition evaluates to false, execution continues with the command following the subroutine return JUMP.

Note that there is only one Return Address register, so subroutine calls cannot be nested. Because the built-in protocols make extensive use of the subroutine call type of jump, a subroutine cannot include a protocol operation.

## 5.6.18.2 Test type

The TEST TYPE field is used to specify when the conditional jump/halt tested condition is considered to be met. The test type options are:

- 00—All specified test conditions are true.
- 01—All specified test conditions are false.
- 10—Any specified test condition is true.
- 11—Any specified test condition is false.

To create an unconditional jump, use TEST TYPE = 00 (all specified conditions true) and clear all TEST CONDITION bits because the tested condition is considered to be true if no test condition bits are set.

To create an unconditional jump/halt with a  $JSL = 1$  conditional wait condition, use  $TEST\ TYPE = 10$  (any specified condition is true). This always jumps or halts once the wait is completed because the selected conditional wait condition(s) are always true after the wait is completed.

A conditional local jump with offset 1 (signifying jump to the following command) is effectively a no-op-except that a wait can be specified, if desired-regardless of the  $TEST\ TYPE$  and  $TEST\ CONDITION$  settings because the next command in sequence is executed whether or not the jump is taken.

### 5.6.18.3 JSL and TEST CONDITION fields

The  $JSL$  field selects between two different interpretations of the  $TEST\ CONDITION$  field:

- When  $JSL = 0$ , the conditional jump/halt bits select various MATH status conditions. These are used to jump or halt if the tested condition is satisfied.
- When  $JSL = 1$ , the bits in the  $TEST\ CONDITION$  field can affect the action taken by the JUMP Command in two ways.
  - Some of the  $TEST\ CONDITION$  bits are conditional jump/halt bits. The  $JQP$ ,  $SHRD$ , and  $SELF$  test conditions are typically used to avoid storing data that the next descriptor might change or to prevent reloading data that is already available because it was left by the previous descriptor.
  - The remainder of the  $TEST\ CONDITION$  bits are conditional wait bits. The  $CALM$ ,  $NIP$ ,  $NIFP$ ,  $NOP$ , and  $NCP$  conditional wait bits are used to time loads, moves, and stores properly. If conditional wait bits are set the JUMP command stalls until all of the specified conditions become true. Then the jump or halt either occurs or not, depending upon whether the tested condition is satisfied. Note that once the wait has completed, the selected conditional wait conditions are always true; because they are evaluated as part of the tested condition, they can affect whether the jump or halt action is taken.

For example, if a JUMP command is executed with  $JSL = 1$  and the  $TEST\ CONDITION$  bits  $NIP$ ,  $NIFP$ ,  $JQP$ , and  $SELF$  are set, the JUMP command stalls until both:

- No input to the input data FIFO is pending ( $NIP$ ).
- No input to the information FIFO is pending ( $NIFP$ ).

Because these are conditional wait bits, the command waits until both conditions are true before evaluating the tested condition. The evaluation depends upon the test conditions that are selected, the state of the selected conditions, and the value in the  $TEST\ TYPE$  field:

## Descriptors and descriptor commands

- TEST TYPE = 00 (if all conditions are true): the jump or halt occurs if another job wants to share this shared descriptor (JQP) and this shared descriptor is running in the same DECO (SELF) as the one from which it was shared.
- TEST TYPE = 01 (if all conditions are false): the jump or halt never occurs because the NIP and NIFP conditions are true after the wait completes.
- TEST TYPE = 10 (if any condition is true): the jump or halt always occurs because the NIP and NIFP conditions are true after the wait completes.
- TEST TYPE = 11 (if any condition is false): the jump or halt occurs if no job wants to share this shared descriptor (JQP) or this shared descriptor is not running in the same DECO (SELF) as the one from which it was shared.

### 5.6.18.4 JUMP command format

**Table 5-56. JUMP command, format**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTYPE = 10100					CLASS		JSL	JUMP TYPE			0	0	0	TEST TYPE	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEST CONDITION								LOCAL OFFSET							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Pointer (see <a href="#">Address pointers</a> ) (Non-local JUMPs only)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pointer (Non-local JUMPs only)															

**Table 5-57. JUMP command, field descriptions**

Field	Description
31-27 CTYPE	Command type 10100 JUMP
26-25 CLASS	Class Wait until specified class type(s) is done before jumping. For CLASS != 00, this makes the JUMP command a DONE checkpoint. 00 None 01 Class 1 10 Class 2 11 Both Class 1 and Class 2
24 JSL	Jump Select Type Selects which definition of the TEST CONDITION field to use. 0 MATH status conditions 1 Various sharing and wait conditions

Table continues on the next page...

**Table 5-57. JUMP command, field descriptions (continued)**

Field	Description
23-21	Jump Type
JUMP TYPE	<p>Specifies the action taken by the JUMP Command. See <a href="#">Jump type</a> for more information.</p> <p>000 Local conditional jump 001 Conditional subroutine call 010 Non-local conditional jump 011 Conditional subroutine return 100 Conditional Halt. This stores the MATH bits as status. (see "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field) 101 Reserved 110 Conditional Halt with user-specified status. This stores the value in the LOCAL OFFSET field as status. 111 Reserved</p>
20-18	Reserved
17-16 TEST TYPE	<p>Test Type. This field defines how the condition code bits (see TEST CONDITION field) should be interpreted. See <a href="#">Test type</a> for more information.</p> <p>00 Jump/halt if ALL selected conditions are true. That is, jump or halt if all the status conditions are true for all TEST CONDITION bits that are 1. Note that if JSL = 1 and one or more conditional wait bits is set, the command waits for all selected conditional wait conditions to be true before the conditional jump/halt conditions are evaluated. The jump/halt then takes place if these conditions are all true.</p> <p>01 Jump/halt if ALL selected conditions are false. That is, jump or halt if all the status conditions are false for all TEST CONDITION bits that are 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and the jump/halt will not take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if all of the selected Conditional Jump/Halt conditions are false.</p> <p>10 Jump/halt if ANY selected condition is true. That is, jump or halt if any status condition is true for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then the jump/halt will take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if any of the selected Conditional Jump/Halt conditions are true.</p> <p>11 Jump/halt if ANY selected condition is false. That is, jump or halt if any status condition is false for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then Tested Condition will be evaluated. Whether a wait occurs or not, the jump/halt will take place if any selected Conditional Jump/Halt condition is false.</p>
15-8 TEST CONDITION	Test Condition. The interpretation of the TEST CONDITION field depends upon the value of the JSL field, a shown in <a href="#">Table 5-58</a> . See <a href="#">JSL</a> and <a href="#">TEST CONDITION fields</a> for more information.
7-0 LOCAL OFFSET	For local jumps this field specifies the address of the JUMP target. This field is ignored for non-local JUMPs. If the LOCAL_OFFSET is 0, the target is the start of the Descriptor Buffer. For non-zero values, the target address is relative to the JUMP Command. That is, the field is interpreted as an 8-bit 2's complement number that is added to the index of the JUMP Command to yield the 32-bit word of the target. For Halt with status, the LOCAL OFFSET will be returned in Descriptor status. This will show up in the Output Job Status as the USTA field. If nonzero on halt with status, an error bit is asserted.
31-0 POINTER	Pointer (32-bit). This field is present only for non-local jumps. This is the address of the Descriptor to which to jump.

**Table 5-58. TEST CONDITION bit settings**

Bit #	TEST CONDITION bits when JSL=0			TEST CONDITION bits when JSL=1		
15	Reserved	Must be 0.	—	JQP	Job Queue Pending. The Job Queue Controller has identified that another job wants to share this Shared Descriptor. This bit can be used to avoid storing data that the next Shared Descriptor would just refetch. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
14	Reserved	Must be 0.	—	SHRD	SHARED. This Shared Descriptor was shared from a previously executed Descriptor. Depending on the type of sharing, this bit can be tested to conditionally jump over commands. For example, if the keys are shared they will already be in the Key Registers so decrypting and placing them in the Key Registers must be skipped. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
13	Reserved	Must be 0.	—	SELF	The SELF bit indicates that this Shared Descriptor is running in the same DECO as the one from which it was shared. Hence, the Shared Descriptor may be able to assume that Context Registers, CHAs and other items are still valid or available. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
12	Reserved	Must be 0.	—	CALM	All pending bus transactions for this DECO, whether internal or external, have completed.	Conditional Wait
11	MATH N	The result is negative.	Conditional Jump/Halt	NIP	No input pending	Conditional Wait
10	MATH Z	The result of a math operation is zero.	Conditional Jump/Halt	NIPP	No iNformation FIFO entries pending	Conditional Wait
9	MATH C	The math operation resulted in a carry or borrow.	Conditional Jump/Halt	NOP	No output pending	Conditional Wait
8	MATH NV	Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.	Conditional Jump/Halt	NCP	No context load pending	Conditional Wait

## 5.6.19 MATH command

The MATH command computes simple mathematical functions. The MATH command sets MATH condition bits that reflect the result of the mathematical operation (see MNV, MN, MC, and MZ). These condition bits can be tested with the JUMP commands, providing CAAM with the flexibility to implement conditional processing constructs, including loops.

Virtually any processing scenario can be implemented using a series of CAAM descriptor commands. MATH commands can be used, for example, to subtract a constant byte length representing a fixed portion of the packet (authentication only header) from the total input data length. Likewise, a descriptor can incorporate a MATH command to write a calculated value to the Variable Sequence Out Length (VSOL) register. As long as the length of a variable length field can be determined with calculation, the MATH command can update the variable length of a sequence.

Length must always be specified in the MATH command, as it determines the size of the arguments used to set the MATH status bits. Note that the LENGTH field is used to mask off results after the MATH operation, not before, so the user must present properly sized data.

**Table 5-59. MATH command, format**

31–27		26	25	24	23–20				19–16
CTYPE = 10101		IFB	NFU	STL	FUNCTION				SRC0
15–12		11–8			7	6	5	4	3–0
SRC1		DEST			0	0	0	0	LEN

**Table 5-60. MATH command, field descriptions**

Field	Description
31-27 CTYPE	Command Type 10101 MATH
26 IFB	Immediate Four Bytes 0 Use full length (LEN) 1 Use four bytes of immediate data even if LEN is 8. This allows 8-byte operations to get a 4-byte immediate value without requiring that value to be padded out to 8 bytes and shortens the descriptor by one word.
25 NFU	No Flag Update 0 Math Flag Updates allowed 1 Math Flag Updates blocked. Preserved the existing flag values by blocking all updates to Math Flags.

*Table continues on the next page...*

**Table 5-60. MATH command, field descriptions (continued)**

Field	Description
24 STL	<p>Stall.</p> <p>0 No stall</p> <p>1 Stall MATH command. Causes the MATH command to take one extra clock cycle. This is not expected to be used by user-defined descriptors (unless there is a need to slow down a loop), but the built-in protocols use this extensively.</p>
23-20 FUNCTION	This field specifies which function to perform. The result is written to the location specified in the DST field.
19-16 SRC0	<p>The SRC0 field indicates the source of operand 0 for the MATH Command.</p> <p>0000 Math Register 0</p> <p>0001 Math Register 1</p> <p>0010 Math Register 2</p> <p>0011 Math Register 3</p> <p>0100 Immediate data. If the data is less than 8 bytes, it is left-extended with 0s. If the data is less than 4 bytes it must be right-aligned. If SRC0 and SRC1 both specify Immediate data, the SRC0 data is in the first word following the MATH command and the SRC1 data is in the second word, and either the LEN field must be set to 4 bytes or the IFB field must be set to 1, or an error is generated.</p> <p>0111 DECO Protocol Override Register, left-extended with 0s.</p> <p>1000 Sequence In Length (SIL), left-extended with 0s.</p> <p>1001 Sequence Out Length (SOL), left-extended with 0s.</p> <p>1010 Variable Sequence In Length (VSIL), left-extended with 0s.</p> <p>1011 Variable Sequence Out Length (VSOL), left-extended with 0s.</p> <p>1100 ZERO (the value 00000000h)</p> <p>1111 ONE (the value 00000001h)<sup>1</sup></p> <p>All other values for this field are reserved.</p>

*Table continues on the next page...*

**Table 5-60. MATH command, field descriptions (continued)**

Field	Description
15-12 SRC1	<p>The SRC1 field indicates the source of operand 1 for the MATH Command.</p> <p>0000 Math Register 0 0001 Math Register 1 0010 Math Register 2 0011 Math Register 3</p> <p>0100 Immediate data. If the data is less than 8 bytes, it will be left-extended with 0s. If the data is less than 4 bytes it must be right-aligned. If SRC0 does not specify Immediate data, the SRC1 data is right-aligned in the first word(s) following the MATH command. If SRC0 and SRC1 both specify Immediate data, the SRC0 data is in the first word following the MATH command and the SRC1 data is in the second word, and either the LEN field must be set to 4 bytes or the IFB field must be set to 1, or an error is generated.</p> <p>0111 DECO Protocol Override Register, left-extended with 0s.</p> <p>1000 Variable Sequence In Length (VSIL), left-extended with 0s.</p> <p>1001 Variable Sequence Out Length (VSOL), left-extended with 0s.</p> <p>1010 Input Data FIFO (the input data FIFO is popped when the MATH command executes unless the function is shld [shift and load]. Note that this means a final pop may have to be done if the data consumed by the shld is the end of the data. If this is the last data to be consumed by DECO, then it is not necessary to pop the data, leaving it there is not a problem.)</p> <p>1011 Output Data FIFO (output data FIFO is popped when the MATH command executes)</p> <p>1100 ONE (the value 0000 0001h)<sup>1</sup></p> <p>1101 Job Source. A code representing the source of the executing descriptor is used as the SRC1 value: 00000000h Job Ring 0 00000001h Job Ring 1</p> <p>1111 ZERO (the value 0000 0000h)<sup>1</sup></p> <p>All other values for this field are reserved.</p> <p><b>NOTE:</b> If SRC1 specifies either input data FIFO or output data FIFO, the MATH command does not execute until the corresponding FIFO has valid data. It is up to the user to ensure that a sufficient amount of data is present. The user must also realize that data comes out of the FIFOs left aligned. This means that if there are only five bytes, the data is in the left 5 bytes, not in the right 5 bytes, of the 8-byte source word. When SRC1 is the input data FIFO, the descriptor must have already created an information FIFO entry to get data to the DECO alignment block, from which the MATH command will pop it.</p>

*Table continues on the next page...*

**Table 5-60. MATH command, field descriptions (continued)**

Field	Description
11-8 DEST	The DEST field specifies the destination for the result of the MATH command:  0000 Math Register 0 0001 Math Register 1 0010 Math Register 2 0011 Math Register 3  1111 DECO Protocol Override Register (Length > 32 bits yields an error) 1000 Sequence In Length (Length > 32 bits yields an error) 1001 Sequence Out Length (Length > 32 bits yields an error) 1010 Variable Sequence In Length (VSIL) (Note that length > 32 bits yields an error) 1011 Variable Sequence Out Length (VSOL) (Note that length > 32 bits yields an error)  1111 No Dest. The result should not be written anywhere. No Dest is useful for setting flags when the actual result is not needed. This selection is illegal if the selected FUNCTION is shift_l or shift_r.  All other values for this field are reserved.
7-4	Reserved. All bits must be 0.
3-0 LEN	LEN denotes the length, in bytes, of the operation and the immediate value, if there is one. All values not specified below are reserved.  0001 1 byte 0010 2 bytes 0100 4 bytes 1000 8 bytes  <b>NOTE:</b> If the selected FUNCTION is shift_l or shift_r, a LEN value other than 1000 may yield unexpected results. Also note that the IFB bit in the command can be used to override the LEN field for the immediate value. When set, the IFB (Immediate Four Bytes) bit allows the MATH command to use a 4-byte immediate value even though it is doing an 8-byte operation.

1. Note that for backward compatibility with earlier versions of CAAM the encoding of this source differs between the SRC1 and SRC0 fields.

**Table 5-61. FUNCTION field values**

Value	Type	Description	Result
0000	add	Perform addition operation on SRC0 and SRC1.	SRC0 + SRC1
0001	add_w_carry	Perform addition with a carry bit operation on SRC0 and SRC1.	SRC0 + SRC1 + C
0010	sub	Perform subtraction operation on SRC0 and SRC1.	SRC0 - SRC1
0011	sub_w_borrow	Perform subtraction with borrow operation on SRC0 and SRC1.	SRC0 - SRC1 - C
0100	or	Perform bitwise OR operation on SRC0 and SRC1	SRC0   SRC1
0101	and	Perform bitwise AND operation on SRC0 and SRC1	SRC0 & SRC1
0110	xor	Perform bitwise XOR operation on SRC0 and SRC1	SRC0 ^ SRC1
0111	shift_l	Perform shift left operation. SRC0 should be shifted by SRC1 bits	SRC0 << SRC1
1000	shift_r	Perform shift right operation. SRC0 should be shifted by SRC1 bits	SRC0 >> SRC1

Table continues on the next page...

**Table 5-61. FUNCTION field values (continued)**

Value	Type	Description	Result
1001	shld	Perform 32-bit left shift of DEST and concatenate with left 32 bits of SRC1. shld is only meaningful when DEST specifies Math Registers 0, 1, 2 or 3. For all other destinations, this function will work like an ADD with SRC0 set to 0. (That is, SRC1 will be placed into DEST.)	{DEST[31:0], SRC1[63:32]}
1010	zbyt	Find zero bytes in SRC0.  {56'h0,!SRC0[63:56],!SRC0[55:48],!SRC0[47:40],!SRC0[39:32], !SRC0[31:24],!SRC0[23:16],!SRC0[15:8],!SRC0[7:0]}	result is shown at left
1011	swap_bytes	Swap the order of the four bytes in the ms half of SRC0, and independently swap the order of the four bytes in the ls half of SRC0.  SRC0[39:32], SRC0[47:40], SRC0[55:48], SRC0[63:56], SRC0[7:0], SRC0[15:8], SRC0[23:16], SRC0[31:24]  If this is used in conjunction with shld, the result of the two MATH operations will be an "end-for-end" swap of all 8 bytes.	result is shown at left

All other values for this field are reserved.

**NOTE:** A Compare operation is accomplished by selecting FUNCTION=sub, with DEST=No Dest.

All MATH commands take one clock cycle to execute except for the shift\_l and shift\_r functions. For the shift\_l and shift\_r functions, the number of bit positions that the data is shifted is specified in the register (or constant) selected by means of SRC1.

- If the data is to be shifted 64 or more bit positions, the shift command takes two clocks. One clock decodes the command, and one clock stores all zeros in the DEST register. Because all bits are shifted off the end, the result is all zeros.
- If the data is to be shifted 63 or fewer bit positions, the shift\_l and shift\_r functions take at most two clocks more than the number of bits in SRC1[5:0] that are 1. The shifter can shift any power-of-2 number of bit positions in one cycle, and there are two additional cycles of overhead. If the intermediate shift result is all 0, the remaining shifts are skipped, resulting in fewer clock cycles than the maximum.

Note that the shift\_l and shift\_r functions first copy the data specified by SRC0 into the register specified by DEST and then shift the data in the DEST register. If the source is 64 bits but the destination is a 32-bit register, the 64-bit source value is truncated to its least-significant 32 bits before the shifting begins. A shift\_l works as expected, but a shift\_r of data from a 64-bit source to a 32-bit destination shifts in 0s rather than shifts in bits from the most-significant 32-bits of the source.

When one source, SRC0 or SRC1, is immediate, then the length may be any legal value. If 1, 2, or 4 bytes, the value is right-aligned in the word following the command. If the value is 8 bytes, then the value is in the two words that follow the command. Note that the immediate data can be 4 bytes even if the LEN is 8 bytes if the IFB bit is set.

## 5.6.20 SEQ IN PTR command

The Sequence In Pointer (SEQ IN PTR) command is used to specify the starting address for an input sequence and the length of that sequence (see [SEQ vs non-SEQ commands](#)). Only one input sequence operation may be active within the DECO at any one time. An input sequence is initiated by executing a SEQ IN PTR command with PRE = 0. This causes the following:

- Starting address of the input sequence to be set to the value in the Pointer field
- The Sequence In Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1).

Note that if the EXT bit is 0, the EXT LENGTH field is omitted from the SEQ IN PTR command.

The input sequence terminates when one of the following occurs:

- All input data is utilized.
- An error occurs.
- A new input sequence is started by executing a SEQ IN PTR command with PRE = 0.

An error is flagged if a SEQ command attempts to input data when the input sequence length is 0 or if the execution of that command would cause the remaining length to go below 0. To extend the length of the sequence any number of additional SEQ IN PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current Sequence In Length register value, but the address for the input sequence is unaffected. In this case the SEQ IN PTR command does not include a Pointer field.

Note that it is possible for the SEQ IN PTR to generate a false overflow error if the PRE bit is set and a large length value is added to a large value already in the Sequence In Length register. If both values are less than 4000 0000h, this false overflow error does not occur. The current value in the Sequence In Length register can be read by means of a MATH command.

If the same input data needs to be processed again, the input pointer can be restored to the original starting address by executing a SEQ IN PTR with RTO = 1. The SEQ IN PTR command does not include a Pointer field in this case.

**Table 5-62. SEQ IN PTR command, format**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
	CTYPE = 11110					0	INL	SGF	PRE	EXT	RTO	RJD	0	0	0	0				
15-0																				
LENGTH (used if EXT = 0)																				
This pointer is omitted if PRE=1 or RTO=1	31-16																			
	Pointer (see <a href="#">Address pointers</a> )																			
	15-0																			
	Pointer																			
This field is omitted if EXT=0	31-16																			
	EXT LENGTH (used if EXT = 1)																			
	15-0																			
	EXT LENGTH (used if EXT = 1)																			

**Table 5-63. SEQ IN PTR command, field descriptions**

Field	Description
31-27 CTYPE	Command Type 11110 SEQ IN PTR
26 0	Reserved
25 INL	In-Line Descriptor. This specifies that a new descriptor is to be found at the start of the data pointed to by the sequence.  An in-line descriptor is found at the start of the data pointed to by the sequence. DECO reads that descriptor (which must not have a shared descriptor) and then executes it. Therefore, a SEQ IN PTR with INL = 1 is the last command that is executed in the current descriptor.  If the INL bit is 1 and the current Input Sequence length is not as large as the in-line descriptor, an error is flagged. Note that it is an error for INL and RJD to both be 1.  See <a href="#">Using in-line descriptors</a> for more information. 0 No in-line descriptor 1 In-line descriptor present.
24 SGF	0 Pointer points to actual data. 1 Pointer points to a scatter/gather table.
23 PRE	Add more length to the previously specified length of the input sequence.  0 The sequence pointer is set to the value of the 32-bit or pointer and the input sequence length is set to the value specified in the LENGTH or EXT LENGTH field.  1 Command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the previous input sequence length.  It is an error for the PRE bit and the RTO bit to both be set.

*Table continues on the next page...*

**Table 5-63. SEQ IN PTR command, field descriptions (continued)**

Field	Description	
22 EXT	Extended Length 0 Input data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command. 1 Input data length value is in the 32-bit EXT LENGTH following the pointer. The 16-bit LENGTH field is ignored.	
21 RTO	Restore. Used to restore an input sequence. 0 Do not restore. 1 Restore. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current Input Sequence length. The original sequence address and RBS and SGF bits are automatically restored. The intended use is to be able to go back to the beginning of a sequence to reprocess some or all of the data. It is an error for the PRE bit and the RTO bit to both be set.	
20 RJD	Replacement Job Descriptor If there is no shared descriptor, this is synonymous with the INL bit (i.e., setting either bit yields the same result). However, if there is a shared descriptor, setting the RJD bit causes the job descriptor to be replaced without affecting the shared descriptor, which will have already been loaded. See <a href="#">Using replacement job descriptors</a> for more information. It is an error if both RJD = 1 and INL = 1. 0 Don't replace job descriptor 1 Replace job descriptor	
19-16 0	Reserved	
15-0 LENGTH	If EXT = 0, LENGTH specifies the number of bytes in (or to be added to) the input sequence. If EXT = 1, LENGTH is ignored	
31-0 POINTER	Pointer Specifies the starting address for an Input Sequence. See <a href="#">Address pointers</a> .	If PRE = 1, this field is omitted.
31-0 EXT LENGTH	If EXT = 0, this field not present. If EXT = 1, EXT LENGTH specifies the number of bytes in (or to be added to) the Input Sequence.	If EXT = 0, this field is omitted.

## 5.6.21 SEQ OUT PTR command

The Sequence Out Pointer (SEQ OUT PTR) command is used to specify the starting address for an output sequence and the length of that sequence (see [SEQ vs non-SEQ commands](#)). Only one output sequence operation may be active within the DECO at any one time.

An output sequence is initiated by executing a SEQ OUT PTR command with PRE = 0. This causes the following:

- The starting address of the output sequence to be set to the value in the Pointer field
- The Sequence Out Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1).

If the EXT bit is 0, the EXT LENGTH field is omitted from the SEQ OUT PTR command.

The output sequence terminates when one of the following occurs:

- An error
- A new output sequence is started by executing a SEQ OUT PTR command with PRE = 0.

To extend the length of the sequence, any number of additional SEQ OUT PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current value in the Sequence Out Length register, but the address for the output sequence is unaffected. In this case, the SEQ OUT PTR command does not include a Pointer field.

It is possible for the SEQ OUT PTR to generate a false overflow error if the PRE bit is set and a large length value is added to a large value already in the Sequence Out Length register. If both values are less than 4000 0000h, this false overflow error does not occur. The current value in the Sequence Out Length register can be read by means of a MATH command.

If the same output data needs to be processed again, the output pointer can be restored to the original starting address by executing a SEQ OUT PTR with RTO = 1. The SEQ OUT PTR command does not include a Pointer field in this case.

**Table 5-64. SEQ OUT PTR command, format**

	31–27	26	25	24	23	22	21	20	19	18	17	16
	CTYPE = 11111	0	0	SGF	PRE	EXT	RTO	0	0	0	0	0
<b>15–0</b>												
LENGTH (used if EXT = 0)												
This pointer is omitted if PRE=1 or RTO=1	31–16											
	Pointer (see <a href="#">Address pointers</a> )											
	15–0											
	Pointer											
This field is omitted if EXT=0	31–16											
	EXT LENGTH (used if EXT = 1)											
	15–0											
	EXT LENGTH (used if EXT = 1)											

**Table 5-65. SEQ OUT PTR command, field descriptions**

Field	Description	
31-27	Command Type.	
CTYPE	11111 SEQ OUT PTR	
26-25	Reserved	
24	0 Pointer points to actual data.	
SGF	1 Pointer points to a scatter/gather table.	
23	Previous. Add more length to the previously specified length of the Output Sequence.	
PRE	0 The sequence pointer is set to the value of the 32-bit pointer and the Output Sequence length is set to the value specified in the LENGTH or EXT LENGTH field. 1 Command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the previous Output Sequence length. Note that it is an error if both PRE = 1 and RTO = 1.	
22	Extended Length	
EXT	0 Output data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command. 1 Output data length value is in the 32-bit EXT LENGTH following the pointer. The 16-bit LENGTH field is ignored.	
21	Restore. Used to restore an Output Sequence.	
RTO	0 Do not restore. 1 Restore. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current Sequence Output Length. The original sequence address and SGF bit are automatically restored. The intended use is to be able to go back to the beginning of a sequence to reprocess some or all of the data. It is an error for the PRE bit and the RTO bit to both be set.	
20-16	Reserved	
15-0	If EXT = 0, LENGTH specifies the number of bytes in (or to be added to) the output sequence.	
LENGTH	If EXT = 1, LENGTH is ignored.	
31-0	Pointer. Specifies the starting address for an Output Sequence. See <a href="#">Address pointers</a> .	If PRE = 1, this field is omitted.
POINTER		
31-0	If EXT = 0, this field not present.	If EXT = 0, this field is omitted.
EXT LENGTH	If EXT = 1, EXT LENGTH specifies the number of bytes in (or to be added to) the output sequence.	

## 5.7 Cryptographic hardware accelerators (CHAs)

This section describes the functionality of each individual CHA used by the DECO.

**Table 5-66. Summary of cryptographic hardware accelerators (CHAs)**

Definition	Abbreviation	What it implements	Cross-reference
ARC-four hardware accelerator	AFHA	The alleged RC4 encryption algorithm	ARC-4 hardware accelerator (AFHA) CHA functionality
Data encryption standard accelerator	DESA	The DES and Triple-DES encryption algorithms	Data encryption standard accelerator (DES) functionality
Random number generator	RNG	A true hardware random number generator and a pseudo-random number generator	Random-number generator (RNG) functionality
Message-digest hardware accelerator	MDHA	The MD-5, SHA-1, SHA-256, SHA-384 and SHA-512 authentication algorithms	Message digest hardware accelerator (MDHA) functionality
AES accelerator	AESA	The AES encryption algorithm	AES accelerator (AESA) functionality

## 5.7.1 ARC-4 hardware accelerator (AFHA) CHA functionality

The AFHA CHA implements the standard ARC-4 cryptography algorithm. It is controlled via the class 1 CHA registers.

### 5.7.1.1 AFHA use of the Mode Register

The AFHA uses the Mode Register as follows:

- The AFHA does not use the ICV or AAI fields of the Mode Register. They should be set to 0 at all times during AFHA operations.
- The Encrypt/Decrypt field of the Mode Register is not used by AFHA. In ARC-4, there is no difference between encrypt and decrypt.
- The Algorithm (ALG) field of the Mode Register must be set to "ARC4" to activate the AFHA.
- The Algorithm State (AS) field of the Mode Register is used to select between the operations described in this table.

**Table 5-67. Mode Register[AS] operation selections in ARC4**

Operation	Description
INIT	The AFHA starts with an Initialization and Permutation of the Sbox before beginning message processing. When the Data Size Register is written, the AFHA begins message processing. At the end of the message processing, the Sbox and context pointers (I and J) may be saved for use in continuing processing.
INIT/FINALIZE	The AFHA starts with an Initialization and Permutation of the Sbox before beginning message processing. When the Data Size Register is written, the AFHA begins message processing. At the end of the message processing, no save of the Sbox nor context pointers can be performed.
FINALIZE	The AFHA starts with the Sbox and context pointers loaded with saved values, for continuation of a message processing. When the Data Size Register is written, the AFHA begins message processing. At the end of the message processing, no save of the Sbox nor context pointers can be performed. In this mode, the Key Register and Key Size registers are ignored.
UPDATE	The AFHA starts with the Sbox and context pointers loaded with saved values, for continuation of a message processing. When the Data Size Register is written, the AFHA begins message processing. At the end of the message processing, the Sbox and context pointers (I and J) may be saved for use in continuing processing. In this mode, the Key Register and Key Size registers are ignored.

### 5.7.1.2 AFHA use of the Context Register

The AFHA uses the Context Register as follows:

- The Context Register is used for loading/saving the I and J pointers only; it is not used as the actual I and J pointers during message processing.
- During loading of the context pointers (in FINALIZE or UPDATE mode), context[0:7] is written to the I pointer, and context[8:15] is written to the J pointer.
- During a save of the context pointers (in INIT or UPDATE mode), the I pointer is written back to context[0:7] and the J pointer is written back to context[8:15].
- No other bits of the Context Register are used by AFHA.

### 5.7.1.3 AFHA use of the Key Register

The AFHA uses the Key Register as follows:

- The Key Register contains the 1-to-16 byte key that is used during permutation in INIT or INIT/FINALIZE modes (the first key byte is from Key Register bits [0:7], the second key byte is from key[8:15], and so on).
- The Key Size Register must be programmed to tell AFHA how many bytes of the Key Register are to be used during permutation.
- Key sizes less than 5 bytes are considered to be a security risk, but the AFHA does allow them.

- Setting the key size to zero, or to a value greater than 16 , produces an AFHA key-size error.
- The Key Register and key size registers are ignored in UPDATE or FINALIZE modes.

### **5.7.1.4 AFHA use of the Data Size Register**

The AFHA uses the Data Size Register as follows:

- The Data Size Register is used to indicate the total size (in bytes) of the message being processed.
- If the Data Size Register is programmed to a value of zero, then the AFHA performs only initialization/permutation, or context save or context restore actions as selected by the Mode Register.

### **5.7.1.5 Save and restore operations in AFHA Sbox and AFHA context data**

If the AFHA is going to process a message in sections in the following way, then the Sbox and context pointers within AFHA must be saved/restored:

1. A piece of the message is to be processed.
2. The state of the AFHA to be saved.
3. That state is later to be restored so that more of the message can be processed.

#### **5.7.1.5.1 What is the Sbox?**

The Sbox is a 256-byte array used during message processing, the contents of which changes with each byte of message being processed.

#### **5.7.1.5.2 What are the I and J context pointers?**

The context pointers I and J are each a single byte of data that changes with each byte of message being processed.

### 5.7.1.5.3 Sbox and context data operations

At the end of an INIT or UPDATE mode operation, the Sbox can be saved to memory (automatically encrypted using the JDKEK or TDKEK), with the data from Sbox being read out in order, first byte to last. If the Sbox is saved, the I and J pointers are saved to the Context Register at the same time, as described in the Context Register section, above.

At the start of UPDATE or FINALIZE mode operations, the Sbox is loaded from memory (automatically decrypted using the JDKEK or TDKEK) in the same order in which it was saved. The I and J pointers are loaded from the Context Register at the same time, as described in the Context Register section, above.

### 5.7.1.6 ARC-4 operation considerations

Consider the following regarding operation of the ARC-4 hardware accelerator:

- After a FINALIZE or INIT/FINALIZE operation completes and the mode register is cleared, the mode may be re-written to INIT or INIT/FINALIZE and a new operation runs, assuming a proper key and data is provided. However, re-writing the mode to UPDATE or FINALIZE generates a mode error.
- If the ARC-4 accelerator receives SBox data from the input-data FIFO when it is not expecting it, it generates a data-sequence error. It does not expect SBox data if it has completed a FINALIZE or INIT/FINALIZE operation (with no reset afterwards, and the Mode Register having been cleared), or if the current mode is INIT or INIT/FINALIZE.
- If loading SBox data into the ARC-4 processor in UPDATE or FINALIZE mode, 256 bytes must be loaded. Attempting to load more or less than 256 bytes generates a data-sequence error.
- The SBox may be loaded before a mode is programmed. However, after loading the SBox, if the mode is programmed to INIT or INIT/FINALIZE, a mode error is generated.
- If UPDATE or FINALIZE mode is written, the SBox must be loaded prior to the mode write. This may have occurred either as a new SBox load, or from having been generated from a previous operation of type INIT or UPDATE.
- The ARC-4 accelerator generates a data-sequence error if any data type other than message data or SBox data is received while it is operating.
- The Sbox data can be stored from ARC-4 only after an UPDATE or INIT operation has completed. Attempting to store the Sbox any other time generates a mode error.
- The ARC-4 accelerator can be used with a data size of 0. If the data size is zero (and the "LAST" bit is set in the iNformation FIFO entry), this allows the ARC-4 to generate the SBox and then indicate DONE.

## 5.7.2 Data encryption standard accelerator (DES) functionality

DES performs encryption and decryption on 64-bit values using the algorithm found in FIPS46-3. The DES module in CAAM supports both single- and triple-DES functionality and ECB, CBC, CFB, and OFB modes as well as key parity checking in compliance with the DES specification. DES is controlled from the class 1 CHA registers.

### 5.7.2.1 DESA use of the Mode Register

The DESA uses the Mode Register as follows:

- The encryption field (ENC) controls whether DESA is encrypting or decrypting data.
- The Algorithm State (AS) field is not used to affect DESA functionality and should be set to zero at all times.
- The Additional Algorithm Information field (AAI) specifies the mode DESA runs. The supported modes are electronic code book (ECB), cipher block chaining (CBC), cipher feedback (CFB-8), and output feedback (OFB), described as follows:
  - ECB mode is a confidentiality mode that features, for a given key, the assignment of a fixed ciphertext block to each plaintext block (analogous to the assignment of code words in a codebook).
  - CBC mode is a confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. CBC mode requires an IV to combine with the first plaintext block. The IV does not need to be secret, but it must be unpredictable.
  - CFB mode is a confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input-block.
  - OFB mode is a confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- Key parity checking for DESA that checks for odd parity within each byte of the key is enabled with a value of 80h in the AAI field.
- The algorithm field (ALG) must be programmed to DES or 3DES.

### 5.7.2.2 DESA use of the Key Register

The DESA uses the Key Register as follows:

- The Key Register contains the 8-, 16-, or 24-byte key that is used during permutation in all DES modes.
- The DES specification defines the key as having odd parity in each byte.
- Key parity can be verified using the correct mode setting.

### 5.7.2.3 DESA use of the Key Size Register

DESA uses the Key Size Register as follows:

- Key size can be either 8, 16, or 24 bytes.
- A key size of 8 is valid only in single-DES mode.
- Values of 16 and 24 bytes can be used only in triple-DES mode.
- An illegal key size error is generated when in single-DES mode with a key size other than 8 or when in triple-DES mode with a key size other than 16 or 24.

### 5.7.2.4 DESA use of the Data Size Register

The DESA uses the Data Size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- All DES modes except OFB expect to process data that is a multiple of 8 bytes and generates an error if the data size written is not an 8-byte multiple.
- This register must be written to start data processing.
- Because writing to the Data Size Register causes the written value to be added to the previous value in the register, the register may be written multiple times while data is being processed in order to increase the amount of input data that will be processed.

### 5.7.2.5 DESA Context Register

The DESA uses the Context Register as follows:

- For CBC, OFB, and CFB modes, the initialization vector is written to and read from the DESA Context Register.

- The value of this register changes as a result of the encryption process and reflects the context of DESA.
- DESA uses the first two words of the Context Register to hold the beginning and final IV value for the CBC, OFB, and CFB modes. The bits are assigned as follows: Context DWord0: IV = desa\_context[ 63:0]

### 5.7.2.6 Save and store operations in DESA context data

DESA is able to process data in chunks by saving the intermediate IV from the Context Register after each chunk of data and restoring the IV and key to the correct registers before processing any subsequent chunks of data.

### 5.7.3 Random-number generator (RNG) functionality

The RNG generates cryptographically-strong, random data. CAAM's RNG utilizes a true random-number generator (TRNG) as well as a pseudo random-number generator (PRNG) to achieve both true randomness and cryptographic strength.

The random numbers generated by the RNG are intended for direct use as secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms. Note that before data can be obtained from the RNG, it must be instantiated in a particular mode by executing the appropriate descriptor. Also, a descriptor must be executed to load the JDKEK, TDKEK and TDSK registers with data from the RNG.

#### 5.7.3.1 RNG features summary

The RNG module includes these distinctive features:

- On-board acceleration of DRBG-HASH (NIST 800-90) using SHA-256.
- The RNG supports the key-generation algorithm defined in the NIST Digital Signature Standard, FIPS Publication 186-2, February 2000.
- Support for two state handles
- Integrated entropy source capable of providing the PRNG with entropy for its seed
- IP slave interface. The RNG may be accessed through an IP slave interface for test purposes.
- IP global interface. The RNG system clock and resets are controlled through the IP global interface.
- IP interrupt interface. The RNG output interrupts are driven on the IP interrupt interface.

- IP test interface. The RNG test signals are controlled via the IP test interface.
- CAAM interface. Interface to supply CAAM with random data as required for descriptors, and to initialize key encryption key and Trusted Descriptor Signing Key Registers
- Random-number stream obtainable via job descriptor
- External interface to furnish random data to other modules such as SNVS

### 5.7.3.2 RNG functional description

While the RNG consists of several, functional sub-modules, its overall functionality can be easily described from the top level in terms of a few functional operations. These operations are seed generation and random number generation. Each of these operations require coordination of the RNG's true random-number generator (TRNG) and pseudo random-number generator (PRNG). The functionality of each sub-module is briefly described in the following subsections. RNG commences its seeding operation automatically following reset.

#### 5.7.3.2.1 RNG state handles

The RNG in CAAM implements two state handles. Each state handle is:

- A completely independent virtual RNG.
- Instantiated independently in deterministic or nondeterministic mode, and with or without prediction resistance
- Seeded (or reseeded) independently with independent entropy

Thereafter, each state handle maintains an independent context for the RNG's deterministic random number generator.

Note that the JDKEK, TDKEK, TDSK and ZMK (if the ZMK is set for hardware programming) and the random data to seed the AES Differential Power Analysis protection (if implemented in the AESA) are initialized by data drawn from State Handle 0, and that any random padding required by CAAM's built-in protocols is also drawn from State Handle 0. Because this data may be confidential, there are special security features to ensure that State Handle 0 is not inadvertently or maliciously instantiated in deterministic mode when it should have been instantiated in nondeterministic mode. See the discussion of the RNGSH0 and RANDDPAR fields in the Security Configuration Register.

### 5.7.3.2.2 RNG NIST certification

CAAM's RNG is designed to be NIST-certifiable. One of the requirements of that certification is the ability to test the RNG prior to normal operation. This requires instantiating the RNG in test (deterministic) mode rather than normal (nondeterministic) operational mode, and then having software run various tests on the RNG. To allow an opportunity for this testing, CAAM does not automatically instantiate the RNG in operational mode or automatically load the JDKEK, TDKEK and TDSK registers, and does not automatically respond to a request for random data from the SNVS. After the tests have completed, or if the tests are going to be skipped, the RNG must be instantiated in operational mode and the JDKEK, TDKEK and TDSK registers must be loaded. These steps are accomplished by executing descriptors as described in this table. The execution of these descriptors must be initiated by software, typically via the job ring interface.

**Table 5-68. RNG descriptors to execute for NIST certification**

Descriptor	Value	Execution
Descriptor to instantiate RNG State Handle 0 in deterministic (test) mode	B080 0004h	HEADER command indicating a descriptor with a length of four 32-bit words
	1281 0004h	LOAD Command with 4 bytes of immediate data; destination is Class 1 Key Size register
	0000 0000h	4 bytes of immediate data (entropy input is null)
	8250 0005h	OPERATION command, Class 1, RNG, Instantiate, Test Mode
Descriptor to uninstantiate RNG State Handle 0	B080 0002h	HEADER command, indicating a descriptor with a length of two 32-bit words
	8250 000Dh	OPERATION command, Class 1, RNG, Uninstantiate, Test Mode
Descriptor to instantiate RNG State Handle 0 in nondeterministic (normal) mode	B080 0002h	HEADER command, indicating a descriptor with a length of two 32-bit words
	8250 0004h	OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode
Descriptor to instantiate RNG State Handle 0 in normal mode and load the JDKEK, TDKEK, and TDSK registers	B080 0006h	HEADER command, indicating a descriptor with a length of six 32-bit words
	8250 0004h	OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode
	A200 0001h	JUMP command, wait until Class 1 (RNG) done then local jump to next command
	1088 0004h	LOAD command, 4 bytes of Immediate data, destination Clear Written Register
	0000 0001h	4 bytes of immediate data (clear the Class 1 Mode Register. This resets the done interrupt and returns the RNG to idle.)
	8250 1000h	OPERATION command, Class 1, RNG, Secure Key, Generate
	The Generate command for secure keys allows for an optional "additional_input" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Generate command.	

1. The Instantiate command allows for an optional "personalization\_string" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Instantiate command.

### 5.7.3.3 RNG operations

RNG operations are performed by appropriately setting the Algorithm State (AS) field of the OPERATION command.

**Table 5-69. RNG operations**

Value of AS	Name	Function
00	State-handle generate operation	Causes the RNG to generate random data from the selected state handle and push that data to the output FIFO. The amount of data generated is based on the value in the Class 1 Data Size register. The descriptor can also provide 256 bits of additional input via the Class 2 Key Register, which is used as additional entropy when generating the requested data. The TST bit value must match the deterministic/nondeterministic mode of the selected state handle, else a test error is generated. A test error is also generated if a Generate command is issued to a state handle that is not instantiated.
01	State-handle instantiation operation	<p>Causes the RNG to set up the initial context for the specified state handle. The state handle remains instantiated in the specified mode (deterministic or nondeterministic) until it is uninstantiated or CAAM is reset. A test error is generated if an attempt is made to instantiate a state handle that is already instantiated.</p> <ul style="list-style-type: none"> <li>• TST bit = 0. Nondeterministic instantiation. When instantiating a state handle in nondeterministic (normal) mode, the state handle is seeded with 384 bits of high-grade random entropy from the TRNG and an optional 256-bit personalization string supplied by the descriptor via the Class 1 Context Register.</li> <li>• TST bit = 1. Deterministic instantiation. When instantiating a state handle in deterministic (test) mode, the state handle is seeded with 256 bits of user-specified entropy supplied via the Class 1 Key register and an additional 128 bits of nonce supplied via the Class 2 Key register. Seeding the state handle with known entropy and nonce values allows for deterministic testing. Note that once the RNGSH0 bit in the Security Configuration register has been set to 1, State Handle 0 can no longer be instantiated in deterministic mode. State Handle 0 produces the random numbers used for nonces and padding within the built-in protocols, so this special protection can be used to prevent accidentally or maliciously substituting a test instantiation in place of a nondeterministic instantiation.</li> </ul>
10	State-handle reseed operation	<p>Causes the RNG to reseed an already instantiated state handle; that is, the current state associated with the selected state handle is replaced with new state information. A test error is generated if an attempt is made to reseed a state handle that is not instantiated.</p> <ul style="list-style-type: none"> <li>• For a state handle in nondeterministic mode, the DRNG is seeded with 384 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register.</li> <li>• For a state handle in deterministic mode, 256 bits of user-specified entropy is taken from the Class 1 Key Register.Nonce is not used for reseeding.</li> </ul>
11	State-handle uninstantiate operation	Causes the RNG to uninstantiate the specified state handle, which prevents the state handle from being used to generate data. The state handle can later be instantiated again. A test error is generated if an attempt is made to uninstantiate a state handle that is not instantiated.

### 5.7.3.4 RNG use of the Key Registers

RNG uses the key registers as follows:

- RNG uses the Class 1 Key Register only when instantiating or reseeding a state handle in deterministic (test) mode.
- RNG uses the Class 2 Key Register only when instantiating a state handle in deterministic (test) mode. In these cases, the descriptor has the TST bit set during the OPERATION command and has loaded known values into the following registers:
  - 256-bit entropy input in the Class 1 Key Register (for instantiate and reseed operations)
  - 128-bit nonce in the Class 2 Key register (only for instantiate operations)
- When instantiating or reseeding a state handle in nondeterministic mode, the key registers are ignored and entropy is instead obtained from the TRNG.

### 5.7.3.5 RNG use of the Context Register

The Class 1 Context register is used to supply an optional 256-bit personalization string when instantiating a state handle in nondeterministic mode, or to supply an optional 256 bits of additional input when reseeding a state handle in nondeterministic mode.

### 5.7.3.6 RNG use of the Data Size Register

The RNG uses the Data Size Register as follows:

- When an RNG generate command is executed, the value in the Data Size Register specifies the number of bytes of random data that should be generated and pushed onto the Output FIFO.
- When an RNG instantiate command is executed, the value in the Data Size Register specifies a reseed interval, measured in clock cycles.
- The RNG uses a default value of 10,000,000 cycles. This means that 10,000,000 generate requests are processed before an automatic reseed operation occurs. For a system with a clock speed between 400 MHz-133 MHz, the reseed happens between 3-20 seconds if RNG operations are being processed at the maximum rate.
- The Data Size Register is a 32-bit value so the user can specify a larger or smaller value. If the user does not specify a reseed interval, the default value is used.

## 5.7.4 Message digest hardware accelerator (MDHA) functionality

The MDHA performs hashing and authentication operations using the hashing algorithms defined in FIPS 180-3 (SHA-1, SHA-224, SHA-256) and MD5. MDHA is controlled by the Class 2 registers.

### 5.7.4.1 MDHA use of the Mode Register

The MDHA uses the Mode Register as follows:

- The Encryption field (ENC) and the Authenticate/Protect (AP) field are not used by the MDHA.
- The ICV field enables ICV checking for MDHA. Starting at the MSB, MDHA verifies the number of bytes in the digest that are defined in the Class 2 ICV Size Register.
- The Algorithm State (AS) field is defined as follows:
  - The Additional Algorithm Information field (AAI) specifies whether Authentication is performed on the data with the specified algorithm. The optional authentication modes are HMAC and HMAC with precomputed IPAD/OPAD.

**Table 5-70. Mode Register[AAI] operation selections in MDHA**

Operation	Description
INIT	The hashing algorithm is initialized with the chaining variables, and hashing begins. Input data must be a multiple <sup>1</sup> of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256.
INIT/FINALIZE	The hashing algorithm is initialized with the chaining variables, and padding is automatically put on the final block of data. Any size of data is supported.
UPDATE	The hashing algorithm begins hashing with an intermediate context and running message length. Input data must be a multiple <sup>1</sup> of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256.
FINALIZE	The hashing algorithms begin hashing with an intermediate context and running message length. Padding is performed on the final block of data. Any size of data is supported.

1 A non-zero multiple for HMAC operations

- The HMAC mode is defined by FIPS 198-1. This can be performed with any of the hashing algorithms.
- The HMAC with precomputed IPAD/OPAD performs the HMAC algorithm but allows the IPAD and OPAD step to be preloaded and started from instead of the KEY.
- The Algorithm field (ALG) must be programmed to MD5, SHA-1, SHA-224, or SHA-256.

## 5.7.4.2 MDHA use of the Key Register

The MDHA uses the Key Register as follows:

- The Key Register is only used when one of the AAI field bits are specified.
- These registers either hold the key or the precomputed IPAD/OPAD split key.
- The size of the IPAD and OPAD are each the size of the digest that is defined by the specified algorithm, except for SHA-224, which is 32 bytes, and SHA-384, which is 64 bytes.

### 5.7.4.2.1 Using the MDHA Key Register with normal keys

When loading the Key Register with the Key Command (See [KEY commands](#)), a KDEST value of 0h results in the key source being loaded, with offset zero, into the Key Register. If the ENC bit = 1 in the Key Command, then the key is decrypted into this register.

### 5.7.4.2.2 Using the MDHA Key Register with IPAD/OPAD "split keys"

The HMAC function uses an HMAC key per the following equation:

$$\text{HMAC}(\text{Key}, \text{Message}) = \text{Hash}[(\text{Key} \oplus \text{OPAD}) \parallel \text{HASH}((\text{Key} \oplus \text{IPAD}) \parallel \text{Message})],$$

where "IPAD" and "OPAD" are constants, "Key" is the HMAC Key, and "HASH" is the chosen hashing function (for example, SHA-256).

#### 5.7.4.2.2.1 Definition and function of IPAD/OPAD split keys

To improve performance, CAAM permits the use of pre-computed IPAD/OPAD "split keys". Computing the values  $\text{Hash}(\text{Key} \oplus \text{IPAD})$  and  $\text{Hash}(\text{Key} \oplus \text{OPAD})$  each require MDHA to perform one block of hashing computation. By performing this computation once, subsequent HMAC computations can save two blocks of a HMAC computation. As a result, the Key Command has an option to signal MDHA that the key being loaded is a precomputed split key.

#### 5.7.4.2.2.2 Process flow of using the Key Register with split keys

When MDHA runs, it turns a key into an IPAD/OPAD pair. MDHA writes this pair back to the Class 2 Key Register. Because the IPAD/OPAD pair is required every time, it saves time to create it once and then reuse it rather than starting with the key again. However, the IPAD/OPAD pair does not appear in the Key Register contiguously. To make the hardware much simpler, the IPAD appears at the start of the Key Register and

the OPAD starts at the midpoint. So, between a 16-byte IPAD and the 16-byte OPAD, there can be 48 bytes of null data. Rather than having to load or store the null data, use the split key type.

#### 5.7.4.2.2.3 Using padding with the split key type to align with storage

When doing a FIFO STORE of the split key type, the user provides a length equivalent to the sum of the bytes in the IPAD and OPAD. That is, in the example above, a total length of 32 bytes. DECO knows to get 16 bytes from the start and another 16 bytes from the midpoint. This saves in encryption time and bandwidth. When loading 20-byte IPAD and OPAD (size would be 40), there must be "padding" of 4 bytes following each. That is: {20 bytes of IPAD, 4 bytes of PAD, 20 bytes of OPAD, 4 bytes of PAD}. The padding can be anything, because CAAM discards it. The reason for this is to make it align with how the encrypted split key is stored, where the extra padding is used to pad each of IPAD and OPAD to an 8-byte boundary so that they can be handled separately.

#### 5.7.4.2.2.4 Length of a split key

Because the split key consists of two blocks of material processed by the selected hash algorithm, the length of a split key is twice the length of the hash algorithm's running digest (note exception below). Storage of the split Key in the Class 2 Key Register, however, is such that the value Hash(Key  $\oplus$  IPAD) is at offset zero, and the value Hash(Key  $\oplus$  OPAD) is at offset 64.

#### 5.7.4.2.2.5 Loading/storing a split key with a key command

A split key may be loaded either in encrypted or in unencrypted form. The DECO command to load a split key is the key command with the KDEST field set to 3h. A split key loaded in this way must be stored contiguously in external memory, and is twice the length of the hash algorithm's running digest. For example, the running digest for SHA-224 is 32 bytes, so the length of a SHA-224 split key in external memory is 64 bytes. In addition, the length for a key command associated with a SHA-224 split key must be 64. A split key that CAAM has generated has also been encrypted, and for SHA-1, has been padded with 8 additional bytes.

#### 5.7.4.2.2.6 Loading/storing a split key with a FIFO STORE command

The DECO command to store a split key is the FIFO STORE command (see [FIFO STORE command](#)), with output-data type set to 16 or 26 (17 or 27 to encrypt using the TDKEK). Generating a split key in this fashion results in the key being encrypted with the JDKEK or TDKEK. Note that the length of an encrypted split key is longer if the FIFO STORE command output-data type selects AES-CCM (16 or 17) for the encrypted

key type. Even if AES-ECB is selected (26 or 27), a SHA-1 encrypted split key is always longer, because CAAM must add 8 bytes of padding before the pre-encrypted 40 bytes of actual split key can be encrypted.

#### 5.7.4.2.2.7 Sizes of split keys

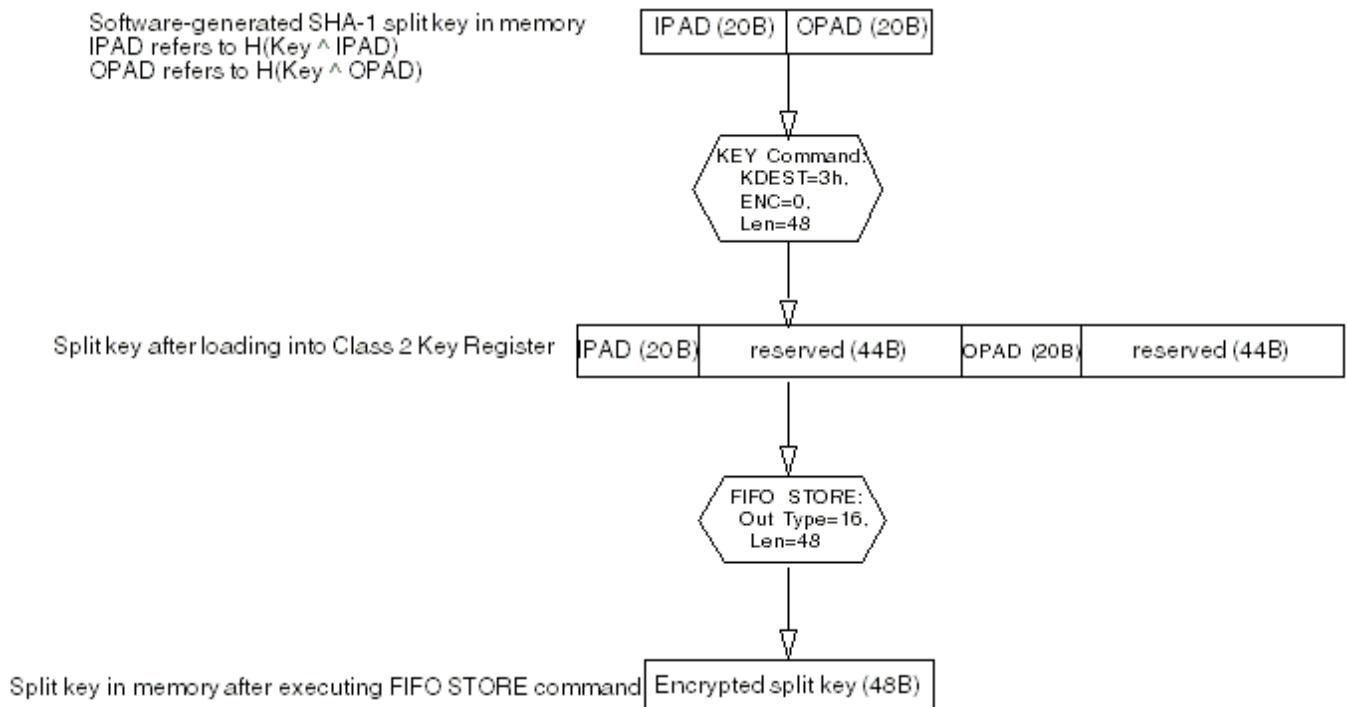
This table describes the different sizes of split keys depending on how they were generated.

**Table 5-71. Sizes of split keys**

Hash algorithm	Final digest size	Running digest size	Software-generated split-key Size	AES-ECB encrypted split-key size	AES-CCM encrypted split-key size
MD5	16 bytes	16 bytes	32 bytes	32 bytes	44 bytes
SHA-1	20 bytes	20 bytes	40 bytes	48 bytes	52 bytes
SHA-224	28 bytes	32 bytes	64 bytes	64 bytes	76 bytes
SHA-256	32 bytes	32 bytes	64 bytes	64 bytes	76 bytes
SHA-384	48 bytes	64 bytes	128 bytes	96 bytes	140 bytes
SHA-512	64 bytes	64 bytes	128 bytes	128 bytes	140 bytes

#### 5.7.4.2.2.8 Constructing an HMAC-SHA-1 split key in memory

This figure is an example of how software would construct an HMAC-SHA-1 split key in memory.



**Figure 5-11. Split keys in memory and in the Class 2 Key Register**

Use the KEY Command to load it into the Class 2 Key Register, and then use the FIFO STORE Command to write it back out in encrypted form.

#### 5.7.4.2.3 MDHA use of the Key Size Register

The Key Size Register is defined to be the number of bytes of key that is loaded into the Key Registers. Key Size ranges are defined as followed:

- MD5: 0 → 64 bytes
- SHA-1: 0 → 64 bytes
- SHA-224: 0 → 64 bytes
- SHA-256: 0 → 64 bytes

#### 5.7.4.3 MDHA use of the Data Size Register

The MDHA uses the Data size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- This register must be written to start data processing.
- This register may be written multiple times while data processing is in progress in order to add the amount written to the register to the previous value in the register.

#### 5.7.4.4 MDHA use of the Context Register

The Context Register stores the current digest and running message length. The running message length will be 8 bytes immediately following the active digest. The digest size is defined as follows:

- MD5: 16 bytes
- SHA-1: 20 bytes
- SHA-224: 28 bytes final digest; 32 bytes running digest
- SHA-256: 32 bytes

#### 5.7.4.5 Save and restore operations in MDHA context data

MDHA is able to process data in chunks by saving the intermediate context and running message length from the Context Register after each chunk of data and restoring the context and running message length to the Context Registers before processing any subsequent chunks of data.

### 5.7.5 AES accelerator (AES) functionality

The advanced encryption standard accelerator (AES) module is a hardware co-processor capable of accelerating the advanced encryption standard (AES) cryptographic algorithm.

#### 5.7.5.1 What is the AES?

AES is a block cipher that processes data in 128-bit blocks. It is a symmetric key algorithm, that is, the "same" key is used for both encryption and decryption, although the key appears in a different form for decryption than it does for encryption.

#### 5.7.5.2 Differences between the encrypt and decrypt keys

The decrypt form of the key is different from the encrypt form of the key, because AES successively modifies the cryptographic key during the steps of the cryptographic operation. The decryption operation yields the correct result only if the modified form of the key (the decrypt key) is used at the beginning of the decryption operation. Unless told otherwise (via the DK bit in the OPERATION command), CAAM assumes that a key

loaded from memory is the encrypt key, that is, the form appropriate for encryption. If a decryption operation is specified and DK = 0, CAAM first goes through the steps required to derive the decrypt key from the encrypt key, and then performs the decryption operation. If a decryption operation is specified and DK = 1, the steps required to derive the decrypt key are skipped and the decryption operation is performed immediately, significantly improving performance for small data blocks.

Note that the difference between the encrypt key and the decrypt key must be taken into account when sharing keys between jobs. When an AES decryption job loads a key from memory, it is probably an encrypt key, so the DK bit in the OPERATION command should be set to 0 so that CAAM derives the decrypt key from the encrypt key before beginning the decryption operation. But when a subsequent AES decryption job shares the key from a previous decryption job, the key that is shared is a decrypt key. In that case, the DK bit should be set to 1, which tells CAAM to skip the key derivation steps. If DK were set to 0 in this case, the decrypt key would be modified as if it were an encrypt key, and consequently, the wrong key value would be used in the decryption operation. Note that a JUMP command with TEST CONDITION set to SHRD (see [Table 5-57](#)) can be used to determine whether the OPERATION command should be executed with DK = 0 or DK = 1.

### 5.7.5.3 AESA modes of operation

The following modes are supported by AESA:

- Electronic codebook (ECB)
- Cipher block chaining (CBC)
- Output feedback (OFB)
- 128-bit cipher feedback (CFB128)
- Counter (CTR)
- Extended cipher block chaining message authentication code (XCBC-MAC)
- Cipher-based MAC (CMAC)
- CTR and CBC-MAC (CCM)
- Combined CBC and CMAC (CBC-CMAC)
- Combined CTR and CMAC (CTR-CMAC)

AES modes can be classified into these categories:

- Confidentiality (ECB, CBC, CTR, OFB, CFB128)
- Authenticated Confidentiality (CCM)
- Authentication (XCBC-MAC, CMAC)

CBC Mode can also be viewed as an authentication mode when used to encrypt data, because it provides CBC-MAC in the context registers.

#### 5.7.5.4 AESA use of registers

Note the following regarding the AESA's use of registers:

- AESA is controlled by the Class 1 registers.
- For all modes, if AES is selected and the mode code written to the Mode Register does not correspond to any of the implemented AES modes, the illegal-mode error is generated.
- In all AES modes, the Key Size Register must be written by the time both the Mode Register and Data Size Register are written. Hence, any sequence of writes to these registers is allowed for sequences in which the Key Size Register is the last to be written. Also, for all AES modes, the bit offset in the Class 1 Data Size Register must be zero when the last write to that register is completed. Failure to comply with these requirements generates the corresponding error in the CCB Status Register.
- When sharing context between consecutive AES jobs, software reset is not issued. To prepare AES for the next job, the Data Size Register and Mode Register must be cleared, as well as the Done Interrupt. The order of these should be such that the Done Interrupt is not cleared first.
- If ICV-only jobs are created (no data to be processed, only ICV to be checked) in modes that support ICV check, the AS mode field should be reset.

#### 5.7.5.5 AESA use of the parity bit

AESA incorporates fault-detection logic based on parity. The parity bit is computed for every byte of input data and key. These parity bits are then fed to the fault detection logic that computes expected parity of every byte for both key and data based on the AES transformations implemented in the main data-path. The expected parity is compared with the parity of the actual key and data bytes and the hardware error is generated if there is a mismatch.

#### 5.7.5.6 AES ECB mode

The electronic codebook (ECB) mode is a confidentiality mode that features, for a given key, the assignment of a fixed, ciphertext block to each plaintext block, analogous to the assignment of code words in a codebook. In ECB encryption, the forward cipher function is applied directly and independently to each block of the plaintext. The resulting sequence of output blocks is the ciphertext. In ECB decryption, the inverse cipher function is applied directly and independently to each block of the ciphertext. The resulting sequence of output blocks is the plaintext.

### 5.7.5.6.1 AES ECB mode use of the Mode Register

AES ECB mode uses the Mode Register as follows:

- The Encrypt (ENC) field should be 1 for ECB encryption and 0 for ECB decryption.
- The ICV/TEST bit is used in ECB mode to activate the fault detection test logic. This logic verifies that the fault detection logic is operational by injecting bit-level errors into input data and key bytes. Because ECB mode does not normally use the Context Registers, the first 128 bits of the context are used in the ECB TEST mode to define which byte of the input data and the key has a bit error injected.
- The Algorithm State (AS) field is not used in ECB mode.
- The Additional Algorithm Information (AAI) field must be set with value 20h that activates ECB mode. Setting the MSB in the AAI field (interpreted as the Decrypt Key or DK bit for AES operations) specifies that the key loaded to the Class 1 Key Register is the decryption form of the key, rather than the encryption form of the key. If DK = 0, when a decryption operation is requested CAAM processes the content of the Class 1 Key Register to yield the decryption form of the key. If DK = 1, CAAM skips this processing. The illegal-mode error is generated if DK = 1 and ENC=1.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 5.7.5.6.2 AES ECB mode use of the Context Register

ECB does not use Context Registers except when fault-detection test is activated. In this case, the first 128 bits of the context are reserved for the error code. The error code:

- Defines which byte of the input data and the key will have a bit error injected.
- Can have 32, 40, or 48 active bits depending on the key size (16, 24 or 32 bytes in ECB mode).
- Is right justified within first 128 bits of the context such that bit 0 of Context DWord 1 injects error into the MSB of the input data, while bit 16 of Context DWord 1 injects error into the MSB of the key.

If all bits of the error code are 0, no error is injected and fault detection logic does not activate the hardware error. If the ICV/TEST bit of the Class 1 Mode Register is 0 in the ECB mode, the Context Registers have no effect on ECB processing.

**Table 5-72. Context usage in ECB mode**

Context DWord	Definition	
	ECB	ECB with ICV/TEST = 1
0	-	ERROR CODE
1	-	

### 5.7.5.6.3 AES ECB Mode use of the Data Size Register

The length of the message to be processed in bytes must be written to the Data Size register. If this value is not divisible by 16, the Data Size error is generated.

### 5.7.5.6.4 AES ECB Mode use of the Key Register

ECB keys must be written to the Class 1 Key Register and can have 16, 24, or 32 bytes.

### 5.7.5.6.5 AES ECB Mode use of the Key Size Register

The number of bytes in the ECB key must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written. Any value other than 16, 24, or 32 causes the key-size error to be generated.

## 5.7.5.7 AES CBC, OFB, CFB128 modes

The CBC, OFB, CFB128 modes are considered together because of their similarities and are described in this table.

**Table 5-73. AES CBC, OFB, CFB128 modes**

Name	Abbreviation	Function
Cipher-block chaining mode	CBC	<p>Confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. The CBC mode requires an IV (Initialization Vector) to combine with the first plaintext block</p> <p><b>NOTE:</b> CBC mode uses both forward and inverse AES cipher. OFB and CFB use only forward AES cipher.</p>
Cipher feedback mode	CFB	<p>Confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input block. AESA implements 128-bit CFB mode where every ciphertext/ plaintext block must have 128 bits</p>
Output feedback mode	OFB	<p>Confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires IV. The last block of OFB input data can have fewer than 16 bytes</p>

### 5.7.5.7.1 AES CBC, OFB, and CFB128 modes use of the Mode Register

The AES CBC, OFB, and CFB128 modes use the Mode Register as follows:

#### Cryptographic hardware accelerators (CHAs)

- The Encrypt (ENC) field should be 1 for encryption and 0 for decryption, except for OFB mode in which this bit is not used.
- The ICV/TEST bit is not used in these modes.
- The Algorithm State (AS) field is used only in CBC mode to prevent IV update in the context for the last data block when set to "Finalize" (2h).
- The Additional Algorithm Information (AAI) field defines which mode is used for processing. For CBC, OFB, and CFB, these values are 10h, 40h, and 30h, respectively. The Decrypt Key [DK] (AAI field MSB) bit affects CBC mode and specifies that the key loaded to the Class 1 Key Register is the decrypt key. The illegal mode error is generated if DK=1 and ENC=1. If the DK bit is set in OFB or CFB128 modes the illegal-mode error is also generated, because these modes do not use inverse AES cipher.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

#### 5.7.5.7.2 AES CBC, OFB, and CFB128 modes use of the Context Register

The AES CBC, OFB, and CFB128 modes use the Context Register as follows:

- CBC, OFB and CFB all use the Context Registers to provide IV, which is updated with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the IV must be saved and later restored for the next chunk to be processed correctly. At the end of CBC processing, IV is also the MAC of the message.
- If the AS field of the Mode Register is set to "Finalize" (2h) in the CBC mode, the last IV update is not written to the context. This enables CBC encryption to effectively perform ECB encryption transformation of a single-block message located in the context in place of IV, and with an all-zero block provided as input data through the FIFO without overwriting the context.

**Table 5-74. Context usage in CBC, OFB, CFB modes**

Context DWord	Definition
0	IV [127:64]
1	IV [63:0]

#### 5.7.5.7.3 AES CBC, OFB, and CFB128 modes use of the Data Size Register

The AES CBC, OFB, and CFB128 modes use the Data Size Register as follows:

- The byte length of the message to be processed must be written to the Data Size Register.

- The first write to this register initiates processing. It can also be written during processing in which case the value written is accumulated to the current state of the register.
- After the Data Size Register is written for the last time, its value must be divisible by 16 in CBC and CFB modes, otherwise the data-size error is generated.
- Only OFB decrements the value in this register with every processed block.

#### **5.7.5.7.4 AES CBC, OFB, and CFB128 modes use of the Key Register**

The AES CBC, OFB, and CFB128 modes use the Key Register as follows:

- A CBC, OFB, or CFB key must be written to the Class 1 Key Register.
- Keys can have 16, 24, or 32 bytes.

#### **5.7.5.7.5 AES CBC, OFB, and CFB128 modes use of the Key Size Register**

The AES CBC, OFB, and CFB128 modes use the Key Size Register as follows:

- The number of bytes in a key must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

#### **5.7.5.8 AES CTR mode**

The counter (CTR) mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. Note that the counter value must be unique for each data block that is encrypted with the same key. CAAM uses a 128-bit counter to ensure that the counter value will not overflow and wrap around.

##### **NOTE**

It is the user's responsibility to ensure that the same key value is not used again following a reset.

#### **5.7.5.8.1 AES CTR mode use of the Mode Register**

The AES CTR mode uses the Mode Register as follows:

- The Additional Algorithm Information (AAI) field should be set to 00h to activate CTR mode. If the Decrypt Key [DK] (AAI field MSB) bit is set, the illegal-mode

error is generated, because CTR uses only forward AES cipher requiring encryption rather than decryption keys.

- The Algorithm State (AS) field when set to "Finalize" (2h) prevents counter update in the context for the last data block.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 5.7.5.8.2 AES CTR mode use of the Context Register

The AES CTR mode uses the Context Register as follows:

- CTR uses context words 2 and 3 to provide initial counter value (CTR0). This value is incremented with the modulo specified in the Mode Register with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the CTR0 field of context has to be saved and later restored for the next chunk to be processed correctly.
- If the AS field of the Mode Register is set to Finalize (2h) in the CTR mode, the last counter update is not written to the context. This enables CTR encryption to effectively perform ECB encryption transformation of a single-block message located in the context words 2 and 3 in place of CTR0 and with all-zero block provided as input data through the FIFO without overwriting the context.

**Table 5-75. Context usage in CTR mode**

Context word	Initial-input definition	Context-switching definition
0	-	-
1	-	-
2	CTR0 [127:64 ]	CTRi [127:64 ]
3	CTR0 [63:0 ]	CTRi [63:0 ]

### 5.7.5.8.3 AES CTR mode use of the Data Size Register

The byte-length of the message to be processed must be written to the Data Size register. The first write to this register initiates processing. It can also be written during processing in which case the value written will be accumulated to the current state of the register. After the Data Size register is written for the last time, the value of this register may not be divisible by 16. CTR decrements the value in this register with every processed block.

### 5.7.5.8.4 AES CTR mode use of the Key Register

- CTR key must be written to the Class 1 Key Register.
- The Key Register can have 16, 24 or 32 bytes.

### 5.7.5.8.5 AES CTR mode use of the Key Size Register

The number of bytes in a key must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written. Any value other than 16, 24, or 32 will cause Key Size error to be generated.

### 5.7.5.9 AES XCBC-MAC and CMAC modes

The AES XCBC-MAC and CMAC modes are described together because of their similarities. They are extensions of the AES CBC mode that produces a key-dependent, one-way hash (or message authentication code (MAC)) in a secure fashion across messages of varying lengths. They also provide data-integrity and data-origin authentication regarding the original message source.

#### 5.7.5.9.1 AES XCBC-MAC and CMAC modes use of the Mode Register

The AES XCBC-MAC and CMAC modes use the Mode Register as follows:

- The Encrypt (ENC) bit is ignored.
- The ICV bit must be set for computed MAC to be compared with the received MAC. The received MAC must be written to the Input Data FIFO after message data and the FIFO data type must be set to ICV. If this bit is not set, XCBC-MAC and CMAC do not expect received ICV to be supplied after message data.
- The Algorithm State (AS) field is defined for XCBC-MAC as shown in this table.

**Table 5-76. Mode Register[AS] operation selections in AES XCBC-MAC**

Operation	Description
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, derived keys K3 and K2 that are XOR-ed with the last message block are computed and stored in the context to be used in the last processing session. The derived key K1 used as an AES key is computed and written back to the Key Register over the original key
INITIALIZE/FINALIZE	Message is processed in a single XCBC session and the final MAC is computed
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
FINALIZE	Message is processed in multiple sessions and the current session is the last one. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. The final MAC is computed

- The Algorithm State (AS) field is defined for CMAC as shown in this table.

**Table 5-77. Mode Register[AS] operation selections in CMAC**

Operation	Function
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, the constant $L = E(K, 0)$ is computed as encrypted block of zeros using key K and stored in the context to be used in the last processing session for derivation of keys K1 and K2. One of these keys will be XOR-ed with the last message block.
INITIALIZE/FINALIZE	Message is processed in a single session and the final MAC is computed
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. The constant L used for key derivation is provided in the context. If decryption is requested, and data size is not written or is set to 0, and ICV bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session
FINALIZE	Message is processed in multiple sessions and the current session is the last one. The constant L used for key derivation is provided in the context. The final MAC is computed

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV bit is set to 1, the illegal-mode error is generated, except for CICV-only jobs.
- The Additional Algorithm Information (AAI) field must be set to 70h for XCBC and 60h for CMAC to be activated. Setting the DK bit (AAI field MSB) will cause the Illegal Mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 5.7.5.9.2 AES XCBC-MAC and CMAC Modes use of the Context Register

The AES XCBC-MAC and CMAC modes use the Context Register as follows:

- No data needs to be provided in the context when starting a new XCBC or CMAC session.
- The computed MAC and the derived keys K2 and K3 are written back to the context by XCBC.
- The computed MAC and the constant  $L = E(K, 0)$ , computed as encrypted block of zeros using key K, are written back to the context by CMAC.
- When a message is split into chunks and processed in multiple sessions, these values need to be saved before context switch and restored before the next chunk of a message is to be processed. At the end of message processing the first 2 words of the context contain the MAC value.

**Table 5-78. Context usage in XCBC-MAC and CMAC modes**

Mode	Context word	Context-switching definition	Final-result definition
XCBC-MAC	0	MAC[127:64 ]	MAC[127:64 ]
	1	MAC[63:0 ]	MAC[63:0 ]
	2	K3[127:64 ]	-
	3	K3[63:0 ]	-
	4	K2[127:64 ]	-
	5	K2[63:0 ]	-
CMAC	0	MAC[127:64 ]	MAC[127:64 ]
	1	MAC[63:0 ]	MAC[63:0 ]
	2	L[127:64 ]	-
	3	L[63:0 ]	-

### 5.7.5.9.3 AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register

The AES XCBC-MAC and CMAC modes use the Class 1 ICV Size Register as follows:

- This Class 1 register is used to provide received ICV/MAC byte-size when it is other than 16 bytes.
- The computed ICV/MAC written to the context in the XCBC mode is always 16 bytes.
- In CMAC mode, this register determines also the computed MAC size-the remaining bytes are cleared.
- Supported values for ICV size are 4, 6, 8, 10, 12, 14 and 16 bytes. If this register is 0, the size of ICV is 16 bytes.

### 5.7.5.9.4 AES XCBC-MAC and CMAC modes use of the Data Size Register

The AES XCBC-MAC and CMAC modes use the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written is accumulated to the current state of the register.
- XCBC-MAC and CMAC decrement the value in this register with every processed block.

### 5.7.5.9.5 AES XCBC-MAC and CMAC modes use of the Key Register

The AES XCBC-MAC and CMAC modes use the Key Register as follows:

- The key must be written to this register.
- For XCBC-MAC, if the AS mode field is set to either "Initialize" or "Initialize/Finalize", it is the original XCBC key (K) that must be written here. Otherwise, the derived key (K1) must be restored to this register. CMAC only uses original key K as an AES key.

### 5.7.5.9.6 AES XCBC-MAC and CMAC modes use of the Key Size Register

The AES XCBC-MAC and CMAC modes use the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- For XCBC-MAC, any value other than 16 causes a key-size error to be generated. For CMAC, this error is generated only if any value other than 16, 24, or 32 is written.

### 5.7.5.9.7 ICV checking in AES XCBC-MAC and CMAC modes

Automatic ICV checking is enabled by setting the ICV bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the illegal-mode error is generated, except for CICV-only (Check-ICV-only) jobs.

The received ICV must be provided on the FIFO after the message data. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is provided in the Class 1 ICV Size register.

If the ICV check detects a mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

### 5.7.5.10 AESA CCM mode

CCM consists of two related processes: generation encryption and decryption verification, which combine two cryptographic primitives: counter mode encryption (CTR) and cipher-block chaining based authentication (CBC-MAC). Only the forward cipher function of the block cipher algorithm is used within these primitives. Note that the counter value must be unique for each data block that is encrypted with the same key. CAAM uses a 128-bit counter to ensure that the counter value does not overflow and wrap around.

**NOTE**

It is the user's responsibility to ensure that the same key value is not used again following a reset.

**5.7.5.10.1 Generation encryption**

A cipher-block chaining is applied to the payload, the associated data (AAD), and the nonce to generate a message authentication code (MAC); then counter mode encryption is applied to the MAC and the payload to transform them into an unreadable form, called the ciphertext. Thus, CCM generation encryption expands the size of the payload by the size of the MAC.

**5.7.5.10.2 Decryption verification**

Counter-mode decryption is applied to the purported ciphertext to recover the MAC and the corresponding payload; then cipher block chaining is applied to the payload, the received associated data, and the received nonce to verify the correctness of the MAC.

In CCM mode, the FIFO data type must be set to message type for message data, while for AAD, either AAD or message type can be used.

**5.7.5.10.3 AES CCM mode use of the Mode Register**

The AES CCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV bit must be set for CCM to compare computed MAC with the received MAC when decryption is requested.
- The received MAC must be written to the input-data FIFO after message data and the FIFO data type must be set to ICV.
- Setting the ICV bit causes the received MAC to be decrypted and compared with the computed MAC.
- The number of MSBs to be compared is defined by the MAC size in the CCM IV ( $B_0$ ) as described in the CCM specification.
- If the AS field is set to FINALIZE, but ICV = 0, AESA does not expect received ICV to be put on the input-data FIFO. In that case, MAC is computed and truncated to the specified size for decryption.
- For encryption, the computed MAC is encrypted and truncated to size. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- If ICV = 1 and the decrypted received MAC do not match computed MAC, the ICV error is generated.
- The Algorithm State (AS) field is defined for CCM as follows:

**Table 5-79. Mode Register[AS] operation selections in AES CCM**

Operation	Description
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, the initial counter CTR0 is encrypted in the CTR mode and the B0 is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B0 and written to the context. This AS setting must be used whenever the first part (or whole) AAD is being processed
INITIALIZE/FINALIZE	Message is processed in a single CCM session and the final MAC is computed and encrypted. The initial counter CTR0 and B0 must be provided in the context
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. All context data is restored from the previous session and the key is written to the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV bit is 1 - AS=UPDATE means that a CICV-only job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, decrypts it and compares it to the computed MAC that is restored with the rest of the context from the previous session
FINALIZE	Message is processed in multiple sessions and the current session is the last one. All context data is restored from the previous session and the key is written to the Key Register. The final MAC is computed and encrypted

- Whenever AS is set to Initialize or Initialize/Finalize, context registers must be zero.
- If the AS field is not set to either Initialize/Finalize or Finalize and the ICV bit is set to 1, the illegal-mode error is generated. This does not apply in case when only ICV check is requested as described for AS = UPDATE.
- The Additional Algorithm Information (AAI) field must be set to 80h for CCM to be activated. The C2K bit is used to select a key register. If C2K = 0, CCM uses the key in the Class 1 Key Register. If C2K = 1, CCM uses the key in the Class 2 Key Register. Setting the DK bit causes the illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

#### 5.7.5.10.4 AES CCM mode use of the Context Register

The AES CCM mode uses the Context Register as follows:

- B0 and the initial counter CTR0 must be provided in the context before the first chunk of the message is to be processed. During initialization, the initial counter CTR0 is encrypted in the CTR mode and B0 (which functions like a CBC-MAC IV in CCM) is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B0 and written to the lower 32 bits of the context word 6.
- If there is AAD, the first block of it defines its size, and that value is decoded and written to the upper 32 bits of context word 6. All of the context data must be restored before the next chunk of the message is to be processed in multi-session processing.
- For CCM encryption, the ICV (encrypted final MAC) is written to context words 4 and 5. For CCM decryption, the ICV (received MAC), which is always encrypted, is

decrypted to words 4 and 5. The final computed MAC is written (in clear) to words 0 and 1.

**Table 5-80. Context usage in CCM mode encryption**

Context DWord	Initial-input definition	Intermediate definition	Final-output definition
0	B0[127:64 ]	-	MAC[127:64 ]
1	B0[63:0 ]	-	MAC[63:0 ]
2	CTR0[127:64 ]	CTR[127:64 ]	-
3	CTR0[63:0 ]	CTR[63:0 ]	-
4	-	E(CTR0)[127:64 ]	E(MAC)[127:64 ]
5	-	E(CTR0)[63:0 <sup>1</sup> ]	E(MAC)[63:0 ]
6	-	AAD size, MAC size; see <a href="#">Table 5-82</a>	-

**Table 5-81. Context usage in CCM mode decryption**

Context DWord	Initial-input definition	Context-switching Definition	Final-result definition
0	B0[127:64 ]	-	MAC[127:64 ]
1	B0[63:0 ]	-	MAC[63:0 ]
2	CTR0[127:64 ]	CTR[127:64 ]	-
3	CTR0[63:0 ]	CTR[63:0 ]	-
4	-	E(CTR0)[127:64 ]	Decrypted Received MAC[127:64 ]
5	-	E(CTR0)[63:0 ]	Decrypted Received MAC[63:0 ]
6	-	AAD size, MAC size	-

**Table 5-82. Format of Context DWord 6 in AES-CCM mode**

Bit 63	Bits 62-48	Bits 47-32	Bits 31-3	Bits 2-0
AAD Presence Flag	0	AAD Size	0	Encoded MAC Size

### 5.7.5.10.5 AES CCM mode use of the Data Size Register

The AES CCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written will be added to the current state of the register.

- CCM decrements the value in this register with every processed block.
- The content of the Data Size register must be divisible by 16 after the last write to it if the AS mode field is set to either "Update" or "Initialize". Otherwise, the data-size error is generated. In other words, message splitting can be done only on a 16-byte boundary.

#### 5.7.5.10.6 AES CCM mode use of the Key Register

CCM key must be written to this register; it is always an encryption key.

#### 5.7.5.10.7 AES CCM mode use of the Key Size Register

The AES CCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

#### 5.7.5.10.8 AES CCM mode use of the ICV check

The AES CCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting the ICV bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"-otherwise the illegal-mode error is generated, unless data size is 0 indicating ICV check is only requested. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input data FIFO after the message data. In CCM, received ICV is always encrypted. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is for CCM encoded in the B0.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

### 5.8 Trust Architecture modules

The CAAM Trust Architecture functions are performed in secure memory and the secure key module.

## 5.8.1 Secure key module functionality

CAAM provides protection of session keys by means of black keys (see [Black keys](#)), integrity protection of CAAM descriptors by means of trusted descriptors (see [Trusted descriptors](#)), and protection of long-term secrets by means of blobs (see [Blobs](#)). All of these protection mechanisms make use of special-purpose cryptographic keys that are managed by CAAM's secure key module.

### 5.8.1.1 Initializing and clearing black and trusted descriptor keys

The secure key module initializes special black key encryption keys (see [Black keys](#)) and trusted descriptor signing keys (see [Trusted descriptors](#)) to random values each time that CAAM powers up and clears these keys when SNVS first enters a FAIL state. The secure key module ensures that only CAAM hardware can use these keys. The values cannot be read or extracted from the chip by any means. However, test values can be written or read by software for debug purposes when CAAM is in non-secure mode, (see [Keys available in non-secure mode](#)).

### 5.8.1.2 Black keys and JDKEK/TDKEK

One special cryptographic key used with the black key mechanism is the 256-bit job descriptor key encryption key (JDKEK) (see [Black keys](#)). When a job descriptor instructs CAAM to store a Key Register into memory, the secure key module first encrypts the content of the Key Register using the JDKEK and then stores the resulting black key into memory. When a job descriptor later references that key, the descriptor identifies the key as a black key, causing the secure key module to decrypt the key using the JDKEK before loading the key into a Key Register. Trusted descriptors can also use the JDKEK, but they are permitted to choose the 256-bit trusted descriptor key encryption key (TDKEK) instead of the JDKEK. Using the TDKEK ensures that only trusted descriptors can use particularly sensitive keys, such as keys that are used to derive session keys. If a TDKEK-encrypted key is embedded as immediate data within a trusted descriptor, this ensures that no other key could be substituted for that particular key.

### 5.8.1.3 Trusted descriptors and TDSK

The secure key module also controls use of the 256-bit trusted descriptor signing key (TDSK) that is used to compute the signature (keyed hash) over trusted descriptors (see [Trusted descriptors](#).). The TDSK is used for verifying the signature whenever a trusted

descriptor is executed. The TDSK is used to sign a descriptor only if the descriptor is executed in a specially privileged job ring, or if a trusted descriptor modifies itself during execution.

### 5.8.1.4 Master key and blobs

The special cryptographic key used for blobs is the 256-bit master key that CAAM receives from SNVS. The secure key module uses this master key to derive keys that are used for blob encryption and decryption when CAAM is in secure mode or trusted mode, but uses a known test key for key derivation when CAAM is in non-secure mode or fail mode.

## 5.8.2 Black keys

CAAM's black key mechanism is intended for protection of user keys against bus snooping while the keys are being written to or read from memory external to the SoC. The black key mechanism automatically encapsulates and decapsulates cryptographic keys on-the-fly in an encrypted data structure called a black key. Before a value is copied from a key register to memory, CAAM automatically encrypts the key as a black key (encrypted key) using as the encryption key the current value in the JDKEKR or TDKEKR. When CAAM is instructed to use a black key as an encryption key, CAAM automatically decrypts the black key and places it directly into a key register before using the decrypted value in the user-specified cryptographic operation.

### 5.8.2.1 Black key encapsulation schemes

CAAM supports two different black key encapsulation schemes, one intended for quick decryption, and another intended for high assurance.

- The quick decryption scheme uses AES-ECB encryption.
- The high-assurance black key scheme uses AES-CCM encryption. The AES-CCM mode is not as fast as AES-ECB mode, but AES-CCM includes an "MAC tag" (integrity check value) that ensures the integrity of the encapsulated key.

### 5.8.2.2 Differences between black and red keys

Differences between black keys and red keys include the following:

- Black keys are encrypted, while red keys are un-encrypted.

- A black key is usually longer than the red key that is encapsulated. ECB encrypted data is a multiple of 16 bytes long, because ECB is a block cipher with a block length of 16 bytes. So if the red key that is to be encapsulated in an ECB-black key is not already a multiple of 16 bytes long, it is padded with zeros to make it a multiple of 16 bytes long before it is encrypted, and the resulting black key is this length.
- A CCM-encrypted black key is always at least 12 bytes longer than the encapsulated red key, because the encapsulation uses a 6-byte nonce and adds a 6-byte ICV. If the key is not already a multiple of 8 bytes long, it is padded as necessary so that it is a multiple of 8 bytes long. The nonce and ICV add another 12 bytes to the length.

### 5.8.2.3 Loading red keys

Red keys can be loaded into Key Registers using either a LOAD command or a KEY command with ENC = 0. But keys cannot be stored from Key Registers back to memory in red form. The only way to store keys back out to memory is in black form. This is accomplished by using the FIFO STORE command with an appropriate OUTPUT DATA TYPE value (see [Table 5-35](#), values 10h-27h).

### 5.8.2.4 Loading black keys

The only way that black keys can be successfully loaded is by using a KEY command with ENC = 1 and the proper setting of the EKT bit. The EKT bit in the KEY command indicates which encryption algorithm (AES-ECB or AES-CCM) should be used to decrypt the key. An ECB-encrypted black key can be successfully loaded only with EKT = 0 (ECB mode), and a CCM-encrypted black key can be successfully loaded only with EKT = 1 (CCM mode).

### 5.8.2.5 Avoiding errors when loading red and black keys

There are ways to unsuccessfully load red and black keys that do not produce error messages, so take special care when loading these keys. Note the following known ways of unsuccessfully loading red and black keys:

- If any type of black key is loaded into a key register using a LOAD command or a KEY command with ENC = 0, no error message is generated.
  - Because these commands instruct CAAM to not perform any decryption when loading the key, this simply places the encrypted form of the black key in the key

register. The only indication that something is wrong is that incorrect results are produced if the key is then used in an encryption or decryption operation.

- No error message is generated if a red key or a CCM-encrypted black key is loaded with a KEY command with ENC = 1 and EKT = 0 (indicating ECB mode), but the wrong value is placed in the key register because the key is decrypted using the wrong mode. Again, the only indication that something is amiss are incorrect results.
- If a red key or an ECB-encrypted black key is loaded using a KEY command with ENC = 1 and EKT = 1 (indicating CCM mode), an error is generated because the CCM-mode ICV check fails.

### 5.8.2.6 Encapsulating and decapsulating black keys

CAAM's key-protection policy imposes restrictions on creating black keys and converting between black key types. When loading a red or black key into a Key Register, it is possible to prohibit the key from being written back out to memory at all. Executing a KEY command with NWB = 1 prohibits writing the key out, whereas NWB = 0 permits the key to be stored to memory as a black key. If a red key is loaded into a key register, it can be stored as either an ECB or CCM-encrypted black key (assuming NWB = 0). But if a black key is loaded into a key register, it can be stored out only as the same type of black key as was loaded, as shown in this figure.

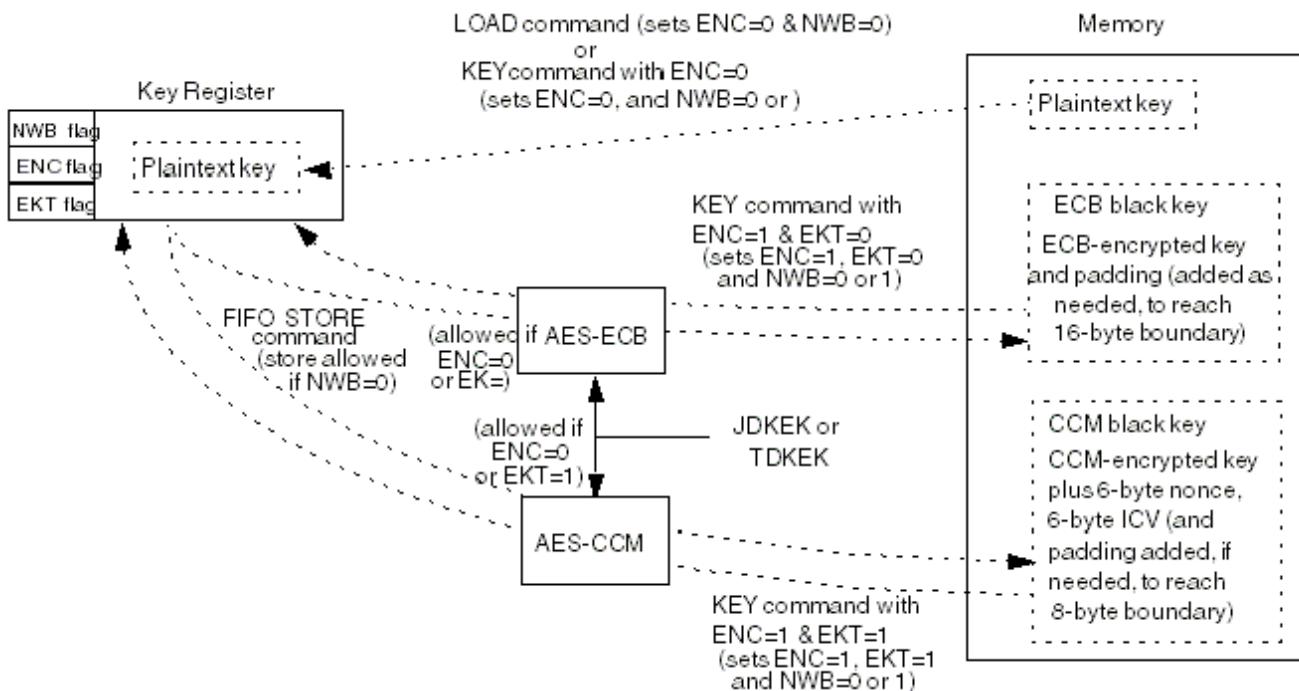


Figure 5-12. Encapsulating and decapsulating CAAM black keys

The cryptographic key used to encrypt or decrypt black keys is held in the 256-bit JDKEKR or TDKEKRJob descriptors or their shared descriptors always use the JDKEKR key, but trusted descriptors, or shared descriptors referenced by trusted descriptors, can use either the JDKEKR key or the TDKEKR key. Use of the TDKEKR allows trusted descriptors to encapsulate keys that cannot be decrypted by job descriptors.

Because black keys are not intended for storage of keys across chip power cycles (CAAM's blob mechanism (section [Blobs](#)) is intended for this purpose), the values in the JDKEKR and TDKEKR are not preserved at chip power-down. Instead, new 256-bit secret values are loaded into the JDKEKR and TDKEKR from the RNG following power-on for use during the current power-on session. That means that a black key created during one power-on session cannot be decrypted on subsequent power-on sessions.

### 5.8.2.7 Types of black keys and their use

The four types of black keys that CAAM's black-key mechanism implements are listed in this table.

**Table 5-83. Black key types**

Black key type	Key used	Encryption mode
JDKEK-ECB	JDKEKR	AES-ECB
TDKEK-ECB	TDKEKR	
JDKEK-CCM	JDKEKR	AES-CCM
TDKEK-CCM	TDKEKR	

Note that it is possible to inadvertently load a black key as the wrong type, resulting in an incorrect key value in the key register. No error message is generated when any of the black key types listed in this table are loaded in ECB mode. But an ICV check failure error message is generated if the wrong black key type (or a red key) is loaded in CCM mode.

It is possible to load a JDKEK-encrypted black key and save it out as a TDKEK-encrypted black key, or vice versa. This is permitted because only trusted descriptors have access to TDKEK encryption, and they are trusted to operate only in a secure manner. Such conversions might be used during a key provisioning procedure. (But as noted earlier, conversion between ECB-black keys and CCM-black keys is not permitted.)

### 5.8.2.8 Types of blobs for key storage

As described in [Blobs](#), CAAM implements different types of blobs that are intended for storage of keys across power cycles. Because encapsulation or decapsulation of blobs takes longer than encapsulation and decapsulation of black keys, if a long-term key is stored in a blob and must be used multiple times during a power-on session, for performance reasons it is preferable to decapsulate the blob at power-up and re-encapsulate the key as a black key.

CAAM implements operations that convert between blob encapsulation and black-key encapsulation without exposing the key in plaintext. There are several different blob types dedicated to key storage that correspond to the different types of black keys. A specific type of black key converts into a specific type of black-key blob. If this were not enforced by CAAM, a hacker could attempt to convert one black key type to another black key type by first exporting the black key as a black key blob, and then re-importing the blob as if it were a different type of black key blob. But because each blob type uses a different derivation for the blob key encryption key, such an attempt at misrepresenting the blob type fails with a MAC-tag error when the blob is decapsulated.

## 5.8.3 Trusted descriptors

Trusted descriptors provide a means for trustworthy software to create trusted "applets" that can be safely executed by less trustworthy software.

### 5.8.3.1 Why trusted descriptors are needed

Software utilizes the cryptographic features of CAAM by building a descriptor, and then adding this descriptor to a job ring. Usually the same software entity performs both operations, that is, building the descriptor and adding it to a job ring. But there are cases in which different software entities perform the two operations. One important case is when the descriptor builder is more trustworthy than the job ring owner. For example, the boot software or TrustZone SecureWorld software might be trusted to properly handle particularly sensitive data, such as digital-rights management keys, but the content-rendering software that needs to use those keys may not be as trustworthy.

CAAM implements a trusted descriptor mechanism to be used in these cases. These trusted descriptors are granted special privileges that ordinary job descriptors are not, and to ensure that these special privileges are not abused by tampering with the trusted descriptor, CAAM ensures the integrity of the trusted descriptor with a cryptographic signature.

### 5.8.3.2 Trusted-descriptor key types and uses

When CAAM is in trusted mode or secure mode, the secure key module ([Secure key module functionality](#)) allows CAAM to use the trusted-descriptor key encryption key and the trusted-descriptor signing key. These keys are available only to CAAM and cannot be read or written. (For testing purposes, these registers, but not the trusted mode or secure mode values of these keys, can be read and written in non-secure mode.) Furthermore, these keys can be used only for key encryption/decryption or signing/signature verification; users cannot use them for anything else. In addition, these keys are changed every boot cycle so that any keys encrypted with the trusted-descriptor key encryption key are lost when the system is rebooted. Likewise, following a reboot, any trusted descriptors signed (HMAC'd) during the previous power-on cycle fail the integrity check and do not execute.

### 5.8.3.3 Trusted descriptors encrypting/decrypting black keys

CAAM implements both trusted and normal (non-trusted) black keys, which are encrypted with different key-encryption keys. Both trusted and normal descriptors are allowed to encrypt or decrypt normal black keys, but only trusted descriptors are allowed to encrypt or decrypt trusted black keys. Note that if any black keys are included as immediate data within the trusted descriptor, it is the encrypted version of the key that is verified when computing the signature. When executing the trusted descriptor, the black key is not decrypted unless the signature is valid.

Trusted software can decapsulate master secrets from trusted-descriptor blobs and can use these master secrets to derive keys that it embeds as trusted black keys within trusted descriptors. Untrusted software can then cause CAAM to execute these trusted descriptors to encrypt or decrypt data, without the master secrets or derived keys ever being directly accessible to the untrusted software.

In addition, trusted descriptors can be written to ensure that these keys cannot be misused. This mechanism would be useful in certain IKE key exchange processes, or for supporting trusted-computing group, trusted-platform module operations, or various data rights-management standards.

See [Black keys](#) for more information.

### 5.8.3.4 Trusted-descriptor blob types and uses

CAAM implements both trusted-descriptor blobs and normal (non-trusted descriptor) blobs, which use different key derivations for the blob-key encryption keys. Both trusted and normal descriptors are allowed to encapsulate or decapsulate normal blobs, but only trusted descriptors are allowed to encapsulate or decapsulate trusted blobs. When executing the trusted descriptor, the blob is not decapsulated unless the integrity check is valid.

See [Blobs](#) for more information.

### 5.8.3.5 Trusted descriptors and secure memory

A secure memory partition can be assigned access-control permissions so that the partition is accessible only by trusted descriptors. Such a partition can be used to store sensitive data, such as cryptographic keys, that can be read or altered only by trusted descriptors. See [Secure memory](#) for more information.

### 5.8.3.6 Configuring the system to create trusted descriptors properly

#### NOTE

Trusted descriptors use the descriptor commands defined in [Using descriptor commands](#). The SIGNATURE command ([SIGNATURE command](#)) is used only by trusted descriptors.

Although trusted descriptors cannot be forged or altered in unauthorized ways after they are generated and signed, to be truly considered "trusted," the system must be configured so that trusted descriptors can be created only by trusted software. Trusted descriptors can be created only via a job ring that has the Allow Make Trusted Descriptor (AMTD) bit set in the job ring's JRMID register. Proper configuration is required to ensure that only trusted software can write to any JRMID register (because this would allow the AMTD bit to be set). This can be ensured in any of the following ways:

- The register is written and then locked (via its LCK bit) by trusted boot software.
- The system ensures that only trusted software runs on the manager processor (because the JRMID registers are writeable only by the manager processor).
- The system uses the operating system or hypervisor to control access to the 4 KB page that includes the JRMID registers.

Proper configuration for the use of trusted descriptors must also ensure control of access to the trusted-descriptor-creation job rings, that is, those job rings whose JRMID registers have been configured with the AMTD bit set. The operating system or hypervisor can

provide access control by granting certain processes access to the 4 KB page containing a particular job ring's control registers, and denying access to that page to other processes. This can also be accomplished by setting the JROWN fields within each JRxMID register to the MID value of a processor and privilege state known to run software with an appropriate level of trust.

### 5.8.3.7 Creating trusted descriptors

To create a trusted descriptor, trustworthy software builds a candidate trusted descriptor that uses the extra privileges properly. For example, the trusted descriptor might utilize cryptographic keys that an ordinary job descriptor cannot access, but the trusted descriptor would be designed so that the key values cannot be exposed.

The candidate trusted-descriptor is converted to a trusted descriptor by executing the candidate trusted-descriptor in a specially-privileged job ring. This causes CAAM to cryptographically sign the descriptor. The trusted descriptor can later be executed by less trustworthy software. When the trusted descriptor is executed, CAAM executes the commands within the trusted descriptor only if the signature is correct. This ensures that the trusted descriptor has not been tampered with after it was created.

#### 5.8.3.7.1 Trusted descriptors and descriptor-header bits

Descriptor headers contain two bits related to trusted descriptors. See [HEADER command](#)" for a full explanation of the descriptor header.

The first bit is the Trusted Descriptor (TD) bit, which when set identifies the descriptor as a trusted descriptor. The second bit is Make Trusted Descriptor (MTD), which instructs CAAM to not execute the descriptor, but to instead turn this descriptor into a trusted descriptor. CAAM does this by first clearing the MTD bit, next computing a HMAC over the Descriptor, and appending that HMAC to the new trusted descriptor. A descriptor with the MTD bit set must be executed from a specially-privileged job ring (AMTD bit set), or an error results.

#### 5.8.3.7.2 Trusted-descriptor execution considerations

Important rules of use and things to consider when executing trusted descriptors are as follows:

- When a trusted descriptor is executed, CAAM first checks the signature (HMAC) to verify that the trusted descriptor has not been modified. If the trusted descriptor references a shared descriptor, it is included in the computation of the signature. If

the signature is valid, the trusted descriptor is executed. If the signature is invalid, the job is aborted with an error indication.

- If a trusted descriptor contains a jump to another descriptor, it must also be trusted. Jumping from a trusted descriptor to a job descriptor results in an error and processing stops. Because all CHAs, all MODEs, and the Class 2 Key and Key Size Registers are reset before a trusted descriptor's signature is checked, care must be taken when transferring to a trusted descriptor from another descriptor (whether trusted or not) via Non-Local JUMP (see Section [JUMP \(HALT\) command](#) ).
- Note that although address pointers within a trusted descriptor are protected against modification, any data referenced by an address pointer is not protected against modification. Therefore, keys and other information that must be protected against modification should be contained as immediate data within the trusted descriptor. When a trusted descriptor executes, it is permitted to modify itself just like a non-trusted descriptor can. This ability can be useful if the trusted descriptor is maintaining an integrity-protected value that changes, such as a usage count, sequence number, and so on. Because modifying the trusted descriptor renders the signature invalid, the signature must be recomputed after the modification. This can be accomplished by placing a SIGNATURE Command at the end of the trusted descriptor. This directs CAAM to recompute the trusted descriptor's signature.

### **5.8.3.7.3 Using master identifier (MID) to access other memory areas**

MID values are asserted by CAAM DMA during descriptor execution. These values are specified in one of the MID registers, depending upon the job source. Job rings can be configured so that descriptors that run in different job rings assert different MID values, allowing each job ring's descriptors to have different memory access privileges. Descriptors can assert either of two MID values, the SEQ MID or the non-SEQ MID.

## **5.8.4 Blobs**

CAAM protects data in a cryptographic data structure called a blob, which provides both confidentiality and integrity protection.

### **5.8.4.1 Blob protocol**

CAAM's built-in blob protocol provides a method for protecting user-defined data across system power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Each time that the blob protocol is used to protect data, a different randomly generated key is used to encrypt the data. This random key is itself encrypted using a key encryption key and the resulting

encrypted key is then stored along with the encrypted data. The key-encryption key is derived from the chip's non-volatile secret key (also known as the "one-time programmable master key" or "OTPMK"), so the key-encryption key can be recreated when the chip powers up again. The combination of encrypted key and encrypted data is called a blob.

[Table 5-51](#) shows the format of the PROTINFO field for the blob protocol, and [Table 5-52](#) describes the bit values.

### 5.8.4.2 Why blobs are needed

To retain data across power cycles, the data must be stored in non-volatile memory. But data stored in this manner is potentially vulnerable to disclosure or modification when the SoC's software and hardware security-mechanisms are not functioning, for example, during debug operations. CAAM is able to protect data for long term storage by encrypting that data with a secure non-volatile key. Using a unique non-volatile key for each device prevents data encrypted on one device from being copied and decrypted on a different device, allowing the secrecy of the data to be compromised.

### 5.8.4.3 Blob conformance considerations

Generation of private blobs is not considered in any governmental security specification. However, there are several steps in the process that can be viewed as having approved methods. These methods were chosen to conform to the following specifications, (except where noted). Assurances that these functions operate as intended are not provided by this document, but must be obtained elsewhere:

- FIPS PUB 197, *Advanced Encryption Standard (AES)*, November 26, 2001.
- FIPS PUB 180-2, SECURE HASH STANDARD, August 1, 2002.
- FIPS PUB 186-2, DIGITAL SIGNATURE STANDARD (DSS), January 27, 2000.
- SP800-38c, *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, May 2004.
- SP800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, March, 2007.
- SP800-57, Recommendation for Key Management - Part 1: General, March, 2007.

In the context of CAAM, a blob is encrypted data that is bound to a specific device by virtue of using a secret non-volatile, device-specific master key. This master key is used only for the purpose of creating and extracting blob data, and the value of this key cannot itself be extracted from a device. To protect data requiring high-security strength, blob creation is performed in hardware using 256-bit security strength (see the note below on

random number generation). AES-256 is used as the encryption algorithm. SHA-256 is used for key derivation (SP800-57 specifies that SHA-256 has 256-bit security strength when used in key derivation).

The random-number generator (RNG) used (see [Random-number generator \(RNG\) functionality](#)) is the FIPS-approved generator described in FIPS186-2 plus change notice Appendix 3.1 (with b=256). It uses the allowed modification when used for the generation of random numbers other than for DSA keys. (Note that the NIST Special Publication SP800-57 indicates that this random number generator has a security strength of 128 bits. As such, it does not meet the newer draft requirements of SP800-90, for the generation of keys longer than 128 bits. )

CAAM blobs provide both confidentiality and integrity protection for the encapsulated data. Because a blob protects both confidentiality and integrity, it may be stored in external long-term storage such as Flash. Counter with cipher block chaining-message authentication code (AES-CCM) is used as the bulk encryption algorithm. Note that the MAC associated with a blob provides integrity protection not only for the encrypted data the blob contains, but also for all intermediate keys used in the creation of a blob.

There may be many different blobs existing at the same time, used for many different purposes, and subject to different security policies. To guarantee that blobs are not inadvertently or intentionally swapped, CAAM encrypts different blobs with different keys. Two mechanisms are used to guarantee that a single key is not used to encrypt unrelated data and to ensure that each key is used to encrypt as little data as possible. One of these mechanisms is random-key generation. Each time that a blob is created, CAAM generates a different, random 256-bit key using CAAM's internal hardware random-number generator (RNG). This blob key is used to encrypt the blob data using AES-CCM, which provides both confidentiality and integrity protection. The second mechanism is key derivation, using a device-unique, non-volatile master key as the key-derivation key. The (volatile) random blob key is encrypted with the non-volatile key derived from the master key, and then stored with the blob so that the blob data can be decrypted during subsequent power-on cycles. Different types of blobs are encrypted using different keys derived from the master key. The derived keys are further differentiated by a key modifier supplied by software, which can be used to guarantee that one blob cannot be inadvertently or maliciously substituted for another blob. Software can use these key modifiers to differentiate specific data, or to prevent replay attacks (the replacement of the current blob with an out-of-date version of the blob).

The master key is used in a key derivation function (KDF) similar to that specified in SP800-56 (sec. 5.8.1). That function includes two parties U and V, who both add information for use in deriving their shared key. Here the derived key is used for storage, and so there is only a single involved party, and hence only one block of public and private information. In the current key-derivation function, only a single iteration is

required, because the size of the derived key is the same as the hash function used. Therefore, the counter is not used. The master key is concatenated with a key modifier (which may be a public or private nonce), an AlgorithmID (the blob type) and a security state indicator (that is, non-secure, secure or trusted). This message is then hashed with SHA-256, and the output is used as a blob-key encryption key.

AES-CCM mode uses a nonce and initial counter value as inputs, along with the key and data. SP800-38c requires that the nonce and counter values be unique across all invocations of AES-CCM under a given key. This requirement is met by virtue of a random key being generated for each blob. Because a key is never used more than once, there are no requirements on the nonce and initial counter value. Therefore, both nonce and initial counter value are fully specified, and the same values are used for all blobs. Blob creation uses the formatting function specified in SP800-38c, Appendix A.

The entire 16-byte MAC is stored along with the encrypted data, to provide a strong assurance of integrity. Note that due to the design of the blobs, the MAC provides integrity protection for the data, blob key and blob-key encryption key.

#### **5.8.4.4 Encapsulating and decapsulating blobs**

When encapsulating a blob, CAAM :

1. Obtains a random blob key (BK) value from the RNG
2. Encrypts the data with that BK
3. Derives a blob-key encryption key (BKEK) from the master key
4. Encrypts the BK using that BKEK

When decapsulating a blob, CAAM :

1. Derives a BKEK from the master key
2. Decrypts the BK using that BKEK
3. Decrypts the data with the BK

#### **5.8.4.5 Blob types**

CAAM supports different types of blobs, and a coded value of the blob type is used as an input to the key-derivation function. This prevents a blob that was exported as one type from being imported as another type because it would decrypt improperly and so would fail the MAC tag check. This table lists the types of blobs that CAAM supports. Note that the type categories are orthogonal, i.e. a blob has one type from each type category. For instance, one blob may be a (normal format/black key/secure state/general memory blob), while another blob may be a (test format/general data/trusted state/secure memory) blob.

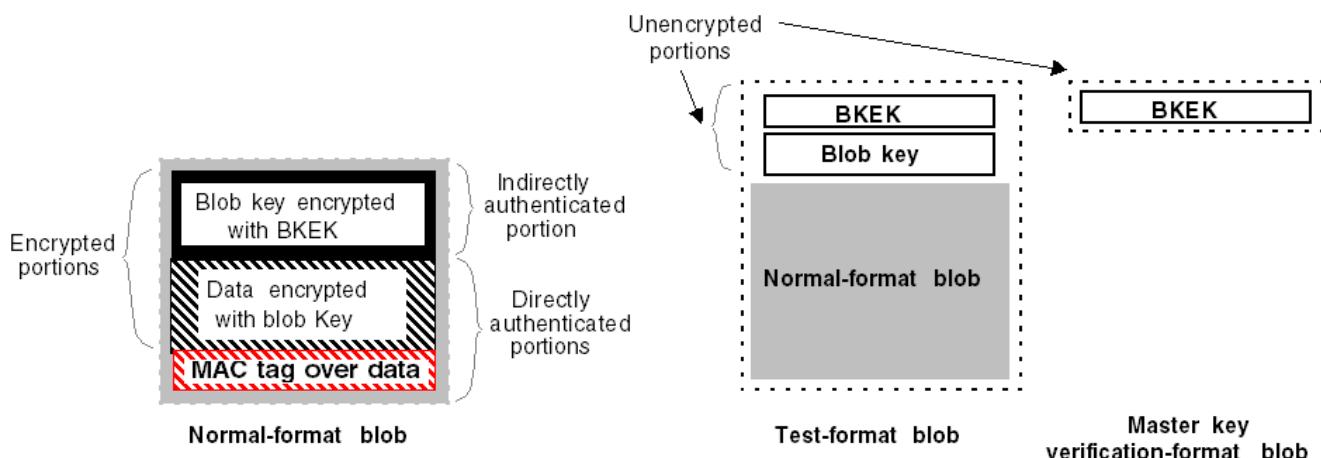
In addition, black key blobs are differentiated by encryption mode and encryption key, so one black key blob may be a (AES-ECB/TDKEK) type and another black key blob may be a (AES-CCM/JDKEK) type.

**Table 5-84. Blob types**

Type Category	Type	Cross-reference
Formats	Normal format	Blob types differentiated by format
	Test format	
	Master key verification format	
Contents	General data (that is, red blobs)	Blob types differentiated by content
	Black keys (that is, black blobs) <ul style="list-style-type: none"> <li>• Encryption modes: AES-ECB and AES-CCM</li> <li>• Encryption keys: JDKEK and TDKEK</li> </ul>	
Security states	Trusted state	Blob types differentiated by security state
	Secure state	
	Non-secure state	
Memory types	General memory	Blob types differentiated by memory type
	Secure memory	

#### 5.8.4.5.1 Blob types differentiated by format

CAAM supports three different formats for blobs, usable for all blob content types, all blob memory types and all blob security state types. This figure describes the blob formats and how they work.



**Figure 5-13. Formats of CAAM blobs**

- A normal-format blob consists of the encrypted blob key, the encrypted data, and a message authentication code (MAC) tag, as shown on the left side of the figure. A randomly-generated, 256-bit blob key is used to encrypt the data using the AES-

CCM cryptographic algorithm. AES-CCM encrypts the data and also yields a MAC tag that is used to protect the data's integrity. The blob key itself is encrypted in AES-ECB mode using a 256-bit blob-key encryption key (BKEK). Checking the MAC directly authenticates the data encapsulated in the blob. The blob key is indirectly authenticated because substitution or corruption of the encrypted blob key yields an incorrect plaintext blob key, which causes the blob content to be decrypted incorrectly, which is detected by the MAC check. Because a normal-format blob is used to protect actual data, the blob-key encryption key (BKEK) that is used to encrypt the blob key for a normal format blob is secret, by virtue of having been derived from the secret master key.

- As shown in the middle of the figure, a test-format blob consists of a normal-format blob, with the unencrypted BKEK and unencrypted blob key prepended. Because the purpose of a test-format blob is to facilitate testing blob encapsulation and decapsulation, the BKEK for a test-format blob is derived from a known test key. CAAM permits test-format blobs to be encapsulated or decapsulated only when CAAM is in non-secure mode.
- As shown on the right side of the figure, a master key verification format blob consists of only the unencrypted BKEK. Because the purpose of a master key verification format blob is to verify that the master key has been properly programmed, the BKEK for a master key verification format blob must be derived from the secret master key. But in order to ensure the secrecy of BKEKs used for normal format blobs, the derivation is different from the derivation used for normal format blobs. This ensures that the BKEKs used to protect data cannot be exposed by examining the BKEK values in master key verification format blobs.

#### 5.8.4.5.2 Blob types differentiated by content

One of the blob content types is intended for general data (see [Red blobs \(for general data\)](#)), and four content types are intended for cryptographic keys (see [Black blobs \(for cryptographic keys\)](#)).

##### 5.8.4.5.2.1 Red blobs (for general data)

Unencrypted data that should be protected is sometimes referred to as "red data", so the type of blob intended for general data (which is left unencrypted when the blob is decapsulated) is called a red blob. When CAAM is instructed to encapsulate data as a red blob, it assumes that the data to be encapsulated is unencrypted and it proceeds to encrypt the data with the blob key. Likewise, when CAAM is instructed to decapsulate a red blob, it assumes that the data that is decapsulated is to be left in memory unencrypted. Other mechanisms, such as an operating system or hypervisor acting in conjunction with a

memory management unit, may be used to protect the data before it is encapsulated into a blob and after it is decapsulated from a blob. CAAM's secure memory can also be used to protect un-encapsulated data.

#### 5.8.4.5.2.2 Black blobs (for cryptographic keys)

CAAM's black blob mechanism is a means for translating between black key encapsulation and blob encapsulation without exposing the key during the translation process. A black blob is simply a blob whose input during blob encapsulation is assumed to be a black key, and whose output during blob decapsulation is either a black key that is written into memory, or an unencrypted key that is placed directly into a Key Register.

CAAM supports the protection of cryptographic session keys by encrypting these keys in a "black key" encapsulation format when storing them in memory via a FIFO STORE command, and then decapsulating them "on-the-fly" as they are referenced by a job descriptor with a descriptor KEY command. Black key encapsulation or decapsulation is very quick, but black keys are intended only for protection during the current SoC power-on session. Black keys encapsulated during one chip power-on session cannot be decapsulated on subsequent power-on sessions because the key encryption key (JDKEK or TDKEK) is erased during power-down and is replaced by a new randomly-generated key encryption key at power-up. To protect a key so that it can be recovered on subsequent power cycles, the key must be encapsulated as a blob. A key could be encapsulated as a red blob, but this would require exposing the key in memory in unencrypted form. To avoid exposing keys in unencrypted form, CAAM supports the concept of black blobs. (Data that is not sensitive to disclosure, either because it is inherently nonsensitive or because it always remains encrypted, is sometimes referred to as "black data".)

#### 5.8.4.5.2.3 Enforcing blob content type

When CAAM is instructed to encapsulate a black blob, it first decapsulates the black key that was specified as input and then encapsulates the resulting key as a Black blob. The black blob itself is exactly the same as a red blob, except that the BKEK derivation is different from red blobs. This prevents a black blob from being decapsulated as a red blob, which would leave the key exposed in memory. Because black keys can be encrypted under either the JDKEK or the TDKEK, and can be encrypted in either AES-ECB mode or AES-CCM mode, CAAM first decrypts the black key data with the appropriate KEK using the appropriate mode and then re-encrypts the key data with the BK using AES-CCM. During this process the key that is temporarily unencrypted is safely protected within CAAM's hardware storage. To prevent mixing up the different types of black blobs (JDKEK vs. TDKEK and ECB vs. CCM), the BKEK for each type is derived differently.

### 5.8.4.5.3 Blob types differentiated by security state

CAAM also supports different types of blobs for use in different security states. All of the blob-format types and blob-content types are available in each of the following different security states:

- Trusted state
- Secure state
- Non-secure state

However, the BKEKs for the blobs are derived differently for each of these states. Therefore, a blob encapsulated while operating in a particular one of these states cannot be decapsulated while CAAM is operating in another of these states:

- During trusted and secure states, the BKEK is derived from the secret master key (but using different key derivation functions in the two states).
- While CAAM is operating in non-secure state, the BKEK is derived from the known test key. This latter type of blob is intended to facilitate testing using known-answer tests.

### 5.8.4.5.4 Blob types differentiated by memory type

CAAM supports blobs for use with different types of memory:

- General memory blobs contain data that originated from any memory accessible to CAAM.
- Secure memory blobs contain data that originated from CAAM's secure memory.

All of the blob-format types, blob-content types, and security-state blob types are available for use with either general memory blobs or secure memory blobs. The input data for secure memory blob encapsulation must all come from a single, secure memory partition, and a secure memory blob can be decapsulated only to a single, secure memory partition, else the encapsulation or decapsulation process is terminated with an error indication before reading or writing the second partition. CAAM immediately aborts any attempt to decapsulate a secure memory blob into memory other than CAAM secure memory, because this would bypass the access controls implemented by secure memory.

#### 5.8.4.5.4.1 General/secure memory blobs and access control

CAAM also prevents the access controls implemented by secure memory from being bypassed by exporting data from a secure memory partition as a blob, and then re-importing the data from the blob into a secure memory partition with different access control permissions. See [Exporting/importing memory type blobs](#) for more information.

#### 5.8.4.5.4.2 Differences between general memory and secure memory blobs

General memory blobs can be used to encapsulate data from, or decapsulate data to, any memory, including secure memory. Secure memory blobs can be used only with secure memory, and all data must be encapsulated from, or decapsulated to, the same partition. Another important difference is that secure memory blobs are subject to different secure memory access control restrictions than are general memory blobs.

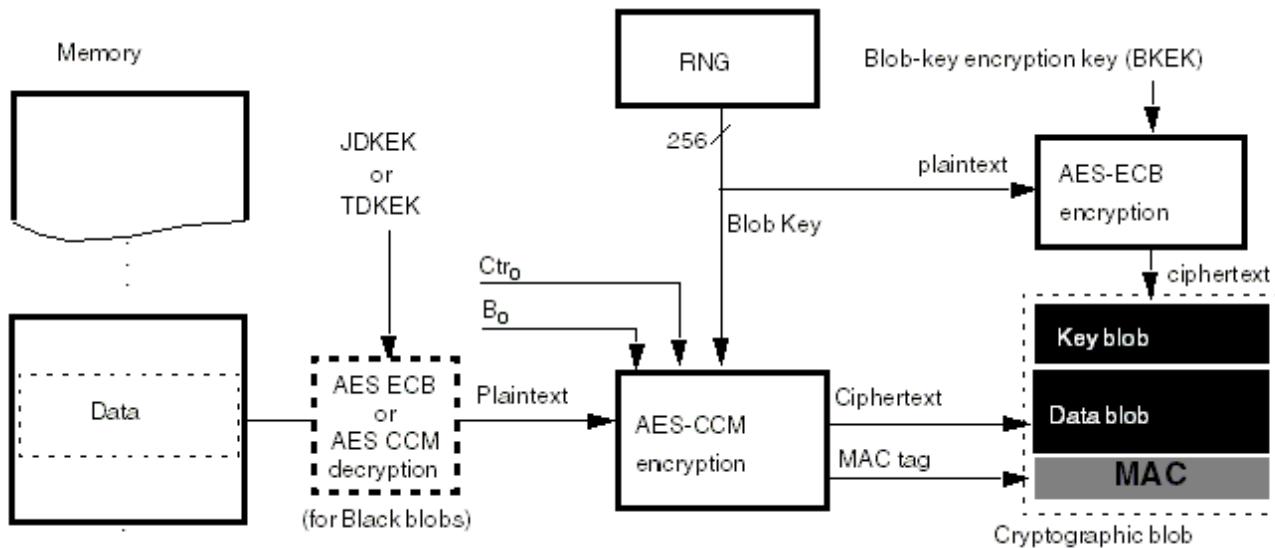
Another important way that secure memory blobs differ from general memory blobs is in the derivation of the blob-key encryption key (BKEK). The BKEK for general memory blobs is derived from the master key (or the test key, in non-secure mode), and the 128-bit key modifier value within the blob descriptor, and a blob type identifier that includes a constant specific to general memory blobs. The BKEK for secure memory blobs is also derived from the master key (or the test key, in non-secure mode), but uses a 64-bit key modifier value within the blob descriptor, and a blob type identifier that includes a constant specific to secure memory blobs, and the values in the SMAPJR and SMAG2/1JR registers of the partition that the blob is being exported from, or imported to. The reason that the SMAPJR and SMAGR contents are included in the BKEK derivation for secure memory blobs is to ensure that the access permissions are cryptographically bound to the blob. This ensures that a secure memory blob can be imported only into a partition with the same access permission settings (and job ring ownership) as the partition from which the blob was exported.

#### 5.8.4.6 Blob encapsulation

A data blob is encrypted using a blob key (BK), which is a random number used as an AES-CCM key. The NIST AES-CCM specification states that for any key, all invocations must use distinct nonces and counter blocks. Although CAAM uses the same nonce and initial counter block values for all data blobs, CAAM satisfies the AES-CCM requirement, because each encryption operation uses a different key (that is, a random number generated by the RNG). The nonce is given as all zeros, and so the initial block  $B0 = 3B00_0000_0000_0000_0000_nnnn\_nnnnh$ , where  $nnnn\_nnnn$  is the number of bytes of plaintext, while the initial counter value  $Ctr0=0300_0000_0000_0000_0000_0000_0000h$ . These values are automatically generated during the encapsulation operation.

[Figure 5-14](#) shows the entire blob-encryption operation. B0 is generated internally and stored in the Class 1 Context DWords 0 and 1, while Ctr0, also generated internally, is stored in Class 1 Context DWords 2 and 3 (see [Table 5-80](#)). The random BK value is stored in the Class 1 Key Register, and the operation mode is set to AES-CCM.

If the plaintext data is in secure memory and the data is exported as a secure memory blob, then all data must be from the same partition. At the blob pointer, the first 32 bytes contain the key blob, which is the encrypted value of the random blob key. Output ciphertext data (data blob) is stored at the blob pointer + 32. The generated message-authentication code (MAC, the signature over the data blob) is stored in the final 16 bytes of the blob.



**Figure 5-14. Encapsulating a blob**

### 5.8.4.7 Blob decapsulation

Before decrypting a data blob, the associated key blob must be decrypted to obtain the blob key. The key blob resides at the blob pointer. AES-ECB mode is used to decrypt the key blob using the BKEK. Generation of the BKEK for general memory blobs and for secure memory blobs is described below.

Ctr0 and B0 are generated internally, and are stored in the Class 1 Context 1 and Context 2 registers, respectively, (see [Blob encapsulation](#)), AES-CCM mode is used to decrypt the data blob (starting at the blob pointer + 32), using the decrypted blob key. If the decrypted data is from a secure memory blob, then the decrypted data must be written into secure memory and all of the data must be written into the same partition. If any of the pages are not within the same partition of secure memory, the blob decryption process is terminated with an error, but any data written prior to the error overwrites the previous data within the partition.

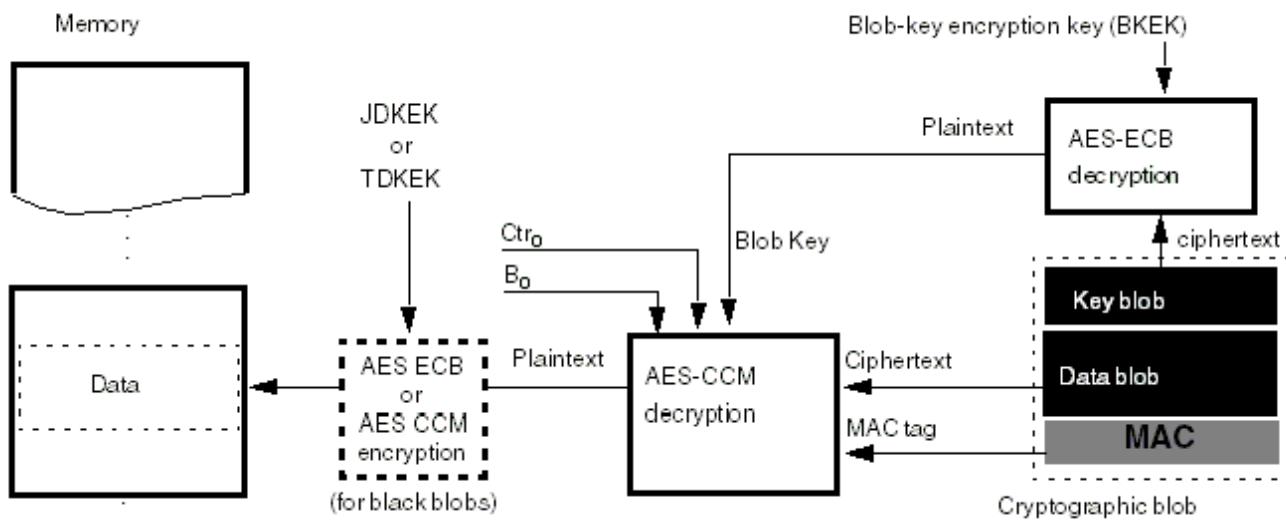


Figure 5-15. Decapsulating a blob

#### 5.8.4.7.1 Operations

Separate the components of the blob:

- Assign  $BK_{ENC} \parallel C \parallel MAC_{16} = Blob\_buffer$
- Encode  $Blob\_Type_{16}$  from the command inputs TK, EKT, K2KR, Sec\_Mem, Black\_key, Blob\_Format and from trusted/job descriptor and the SNVS security state.
- Deriving the BKEK for a general memory blob:
  - $BKEK = SHA-256(Master\_key_{256} \parallel Key\_Modifier_{128} \parallel Blob\_Type_{16})$
- Deriving the BKEK for a secure memory blob:
  - $BKEK = SHA-256(Master\_key_{256} \parallel Partition\_Info_{96} \parallel Key\_Modifier_{64} \parallel Blob\_Type_{16})$  (note that Partition\_Info96 is taken from the SMAP, SMAG2 and SMAG1 registers of the selected partition)
- Decrypt the blob key
- Assign  $BK = AES-ECB-DEC(BKEK, BK_{ENC})$
- Decrypt the plaintext with the blob key, using AES CCM mode
- Assign  $B_0 = 3B00_0000_0000_0000_0000_{xxxx\_xxxx}h$ , where xxxx\_xxxx is the length of the plaintext
- Assign  $Ctr_0 = 0300_0000_0000_0000_0000_0000_0000h$
- Assign  $P = AES-CCM-DEC(BK, B_0, Ctr_0, C, MAC_{16})$
- Assign  $Blob\_buffer = BKEK_{opt} \parallel BK_{opt} \parallel BK_{ENC} \parallel C \parallel MAC_{16}$
- If  $MAC_{16}$  does not match the calculated MAC then return an error
- For a Black\_key blob, encrypt the input under the appropriate KEK (either the JDKEK or TDKEK):

- Assign  $B_{ENC} = \text{AES-ECB-ENC(KEK, P)}$
- or
- Assign  $B_{ENC} = \text{AES-CCM-ENC(KEK, P)}$

### 5.8.5 Critical security parameters

CAAM contains several encryption and authentication keys that are identified as being critical security parameters (CSPs), as defined in FIPS140-2. Each of these CSPs are zeroized (cleared) upon the SNVS state machine entering the FAIL state. This FAIL state indicator is an input to CAAM and can be observed via the CAAM Status Register (see the CAAM Status Register (CSTA)).

Upon receiving an indication that the security state machine has entered the FAIL state, all register-based CSPs are zeroized via the asynchronous hardware reset. Memory-based CSPs are zeroized by means of a hardware-based state machine that writes a constant value to every word of the CSP partitions of the CAAM secure memory. CAAM can be restarted after the chip has transitioned from FAIL to non-secure state; however, all critical security parameters are lost forever.

This table lists the critical security parameters included in CAAM.

**Table 5-85. Critical security parameters**

CSP	Notes	Related cross-reference
Zeroizable secret key	Inside of security power island; loaded and locked once at provisioning time	—
CCB Class 1 Key Register	—	See register appendix
CCB Class 2 Key Register	—	See register appendix
Trusted descriptor signing key	Loaded at boot time from RNG	<a href="#">Trusted descriptors</a>
Trusted descriptor key encryption key		
Job descriptor key encryption key	Loaded at boot time from RNG	<a href="#">Keys available in different security modes</a>
ARC4 SBox	—	<a href="#">ARC-4 hardware accelerator (AFHA) CHA functionality</a>
Crypto-engine internal datapath registers	—	See register appendix
Secure Memory CSP partitions	—	<a href="#">Secure memory</a>
Output data FIFO	—	—

## 5.8.6 Secure memory

The CAAM secure memory can be used either as general-purpose memory for storing data and software, or as special, confidentiality-preserving memory that protects disclosure-sensitive data such as cryptographic keys, passwords, proprietary software, or PIN numbers, or different portions can be used for each purpose.

### 5.8.6.1 CAAM secure memory features

This table lists the features included in CAAM secure memory.

**Table 5-86. CAAM secure memory features**

Feature Type	Feature	Function
Security	Automatic RAM zeroization	Upon detection of a security failure, specified portions of the RAM automatically zeroize
	Access control	Read/write permissions can be set to maintain privacy and integrity of data
	Cryptographic protection of exported data	CAAM can cryptographically protect the confidentiality, integrity and access control permissions of sensitive data exported from the secure memory
Performance/Reuse	AXI slave interface	Supports all bursting modes
	Parameterized for number of partitions, number of pages, and page size	Supports different configurations of secure memory

### 5.8.6.2 Secure memory controller (SMC) states

The secure memory controller (SMC) has the following operational states (corresponding to the non-reset states indicated by the Secure Memory Status Register):

- [SMC initialize state](#)
- [SMC normal state](#)
- [SMC fail state](#)

This figure illustrates the transitions between the states of the secure memory controller.

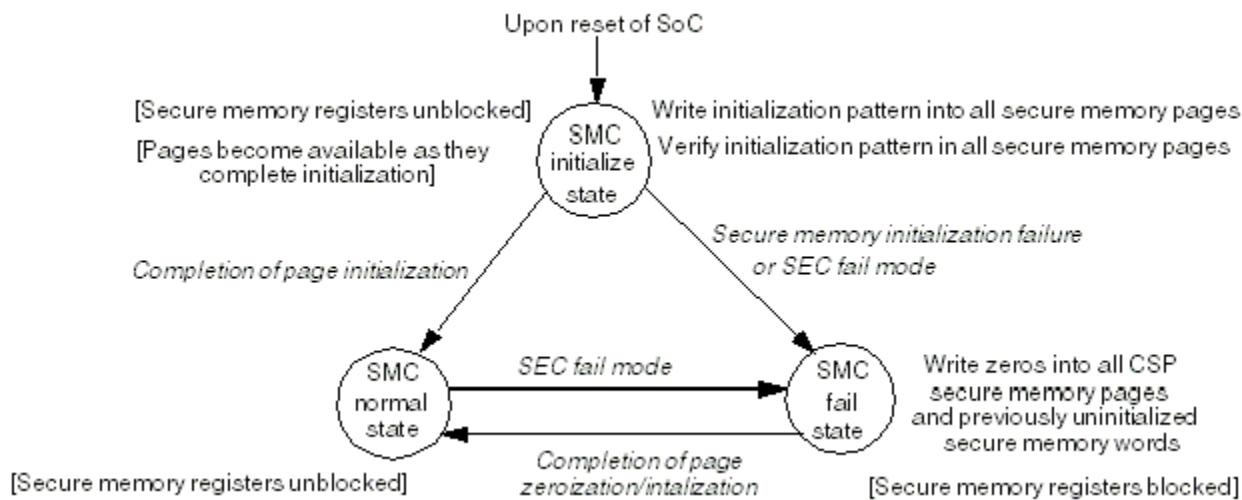


Figure 5-16. Secure memory state machine diagram

### 5.8.6.2.1 SMC initialize state

Upon reset, the secure memory controller begins initializing its memory pages by writing a bit pattern to each memory word in the page. Each word is written with its own address (relative to the beginning address of secure memory) divided by 16. After an entire page has been written, each word of the page is read to verify that the initialization of that page was successful. If initialization was not successful, the secure memory controller sends the CAAM security violation signal to the SNVS and transitions to SMC fail state.

Secure memory also transitions to SMC fail state if CAAM enters CAAM fail mode due a signal from SNVS. Note that asserting the CAAM security violation may cause the SNVS to transition CAAM to CAAM fail mode, depending upon the current register settings and fuse configuration of SNVS.

In SMC initialize state, access is blocked to all pages that have not yet been initialized. As individual pages successfully complete initialization, they are assigned to partition 0 so they may be used while the remaining pages are still being initialized. The PAGE field in the Secure Memory Status Register indicates how many pages are currently available (that is, have completed power-on-reset initialization). Partition 0 is initialized with all access types permitted and all master identities allowed so that by default secure memory may be used as normal RAM. The secure memory registers are not blocked in SMC initialize state, so it is possible to allocate additional partitions, change access permissions, and de-allocate and re-allocate available pages even before the secure memory controller enters SMC normal state.

### 5.8.6.2.2 SMC normal state

The secure memory controller enters SMC normal state from SMC initialize state after successful completion of initialization, and enters SMC normal state from SMC fail state after all CSP pages and previously uninitialized words have been zeroized (that is, overwritten with all 0s). (CSP pages are pages that are currently assigned to any partition that has the CSP bit set in its SMAPregister.)

The secure memory controller remains in SMC normal state unless CAAM enters CAAM fail mode. Note that if SMC normal state was entered from SMC fail state, the secure memory controller immediately transitions back to SMC fail state if CAAM is still in CAAM fail mode.

At POR, partition 0 is automatically allocated to the owner of job ring 0, but by default partition 0 can be accessed using any MID. Processors operating with the job ring 0 owner's MID can alter the permissions for partition 0, or can de-allocate individual pages from Partition 0 or can simply de-allocate partition 0 in its entirety so that pages are available for allocation to other partitions. Additional partitions can be allocated to any job ring, and their access permissions can be changed by the job ring's owner. Note that access is blocked to pages allocated to a partition for which the accessor does not have access permission.

### 5.8.6.2.3 SMC fail state

The secure memory controller enters SMC fail state from either initialize state or normal state when CAAM enters CAAM fail mode. The controller also enters SMC fail state from initialize state if an initialization failure occurs (that is, the controller cannot successfully read the initialization pattern from a word).

While in SMC fail state, the secure memory controller zeroizes (writes an all-0 pattern into) all words that were not already initialized while in SMC initialize state and all words within CSP pages. CSP pages are those pages that are currently allocated to CSP partitions, that is, partitions that have the Critical Security Parameter (CSP) bit set in the partition's SMAP register.

While the secure memory controller is in SMC fail state, the secure memory registers are blocked (that is, they are neither readable nor writable), so no partitions or pages can be allocated or deallocated while in SMC fail state. All non-CSP pages that were already successfully initialized are still accessible, and all other already allocated pages become accessible as they complete their zeroization. The current access permissions remain in effect as long as the secure memory controller remains in SMC fail state. The secure memory controller exits SMC fail state when it has finished zeroizing CSP pages and previously uninitialized secure memory words.

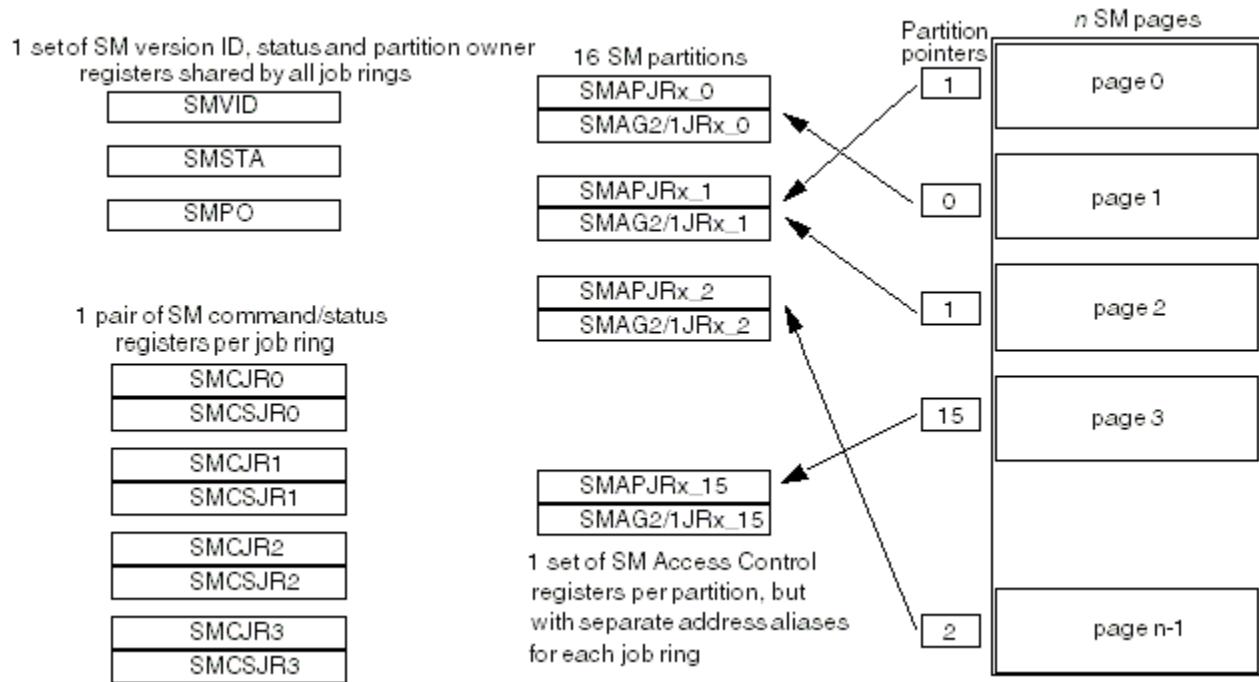
Software can attempt to transition CAAM from CAAM fail mode to CAAM non-secure mode by writing to the appropriate SNVS register. However, the SNVS's security state machine immediately returns to the SNVS fail state if the security-violation input that put the state machine into the fail state remains asserted and unmasked. Note that a secure memory initialization failure causes CAAM's security violation signal to remain asserted. If CAAM is successfully transitioned to CAAM non-secure mode, the secure memory controller returns to SMC normal state and normal operation after it has completed its zeroizations.

### 5.8.6.3 Secure memory organization

As shown in [Figure 5-16](#), CAAM secure memory is divided into access-controlled partitions. The number of implemented partitions can be read from the Secure Memory Version ID Register. An unallocated partition may be claimed by any one of the job ring owners. A job ring owner is identified by the bus MID asserted during its bus transactions. The secure memory aborts accesses to partitions if these accesses are not in accordance with the access permissions set by the partition's current owner.

The secure memory is also divided into equal-sized memory pages. The size and the number of pages can be read from the Secure Memory Version ID Register. Each partition may have any number of the secure memory's memory pages associated with it, thus providing variable-sized partitions.

Each page can be associated with only one partition at a time, and each page has a partition pointer that specifies the partition it is currently associated with. Note that there is no direct indication of which pages are currently associated with a particular partition, although this information can be obtained by issuing a page-inquiry command for each individual page. All pages assigned to a given partition inherit the access permissions set for that partition. When a memory access is started, the partition indicator is used to determine which partition's access control information should apply to the current memory access. This incurs an extra wait cycle to determine the associated partition, but is only incurred during the first read or write of a burst.

**Figure 5-17. Secure memory registers and memory organization**

When a partition is de-allocated, all pages assigned to the partition are freed and may subsequently be assigned to any currently allocated partition. Only the owner of a partition may de-allocate that partition, or assign pages to, or free pages from, that partition. For those partitions that have the Critical Security Parameter (CSP) bit set, CAAM zeroizes all pages allocated to the partition before they are freed and made available for reassignment.

If software loses track of the partition to which a page is assigned, it can re-acquire the information by issuing an inquire-page command via the Secure Memory Command Register and then reading the partition number from the Secure Memory Command Status Register.

### 5.8.6.4 Secure memory security functions

The secure memory's primary security functions are as follows:

- Automatic RAM zeroization
- Access control
- Cryptographic protection of exported data

### 5.8.6.4.1 Automatic RAM zeroization

When CAAM initializes following a power-on reset (POR), the CAAM secure memory controller begins zeroizing the secure RAM by overwriting each word. because there may be sensitive data that remains in the secure memory from before the POR. This could happen either because the POR occurred without an actual loss of chip power, or because the RAM used for the secure memory exhibits some state retention even when powered off. The value that CAAM writes into each word of RAM is not actually zero because CAAM writes a different value to different memory words. This allows the secure memory controller to verify that each word was overwritten by re-reading the word and comparing the read value with the proper value. If there is a mismatch, the secure memory controller raises a security alarm that causes CAAM to go into Fail Mode.

#### 5.8.6.4.1.1 Zeroizing secure memory marked "CSP"

When portions of secure memory that are marked "CSP" are released by the current owner for use by another owner, the secure memory zeroizes (by writing zeros) these memory areas so that sensitive data cannot be scavenged by the new owner. When CAAM detects an internal security error, or enters fail mode because SNVS has detected a security violation, the secure memory immediately blocks access to and begins zeroizing (by writing zeros) the portions of the secure memory that hold sensitive data. The secure memory can identify these portions of memory because the partitions were previously marked by software as CSP (critical security parameter). If a partition is not marked as CSP, this indicates that the partition is not used to store disclosure-sensitive data, so the secure memory does not zeroize that memory area when a security alarm occurs or when that partition or pages from that partition are de-allocated.

### 5.8.6.4.2 Access control

An OS running on a processor uses that processor's MMU to control the accesses that the processor makes to memory. This can be used to protect the OS's memory spaces from user processes running on that processor, and to protect process memory spaces from other processes on that processor. However, a processor's MMU cannot exert any control over accesses made from another bus master, including another processor. For this reason, the secure memory enforces an additional level of access control over the secure memory's internal RAM.

### 5.8.6.4.2.1 Access control in secure memory RAM partitions

The RAM within the secure memory is divided into multiple partitions, each with its own set of access control permissions. The secure memory enforces control over access to partitions by verifying that the bus signals associated with each access are in agreement with the access control permissions for that partition. If they are not in agreement the access is aborted with a bus error.

### 5.8.6.4.2.2 Access control through job rings

When CAAM job descriptors are fetched or executed, CAAM's own DMA engine can access secure memory via a datapath internal to CAAM, so it does not pass through a system MMU, if there is one. Consequently, CAAM is designed to enforce its own access control over accesses from job descriptors to secure memory. Because each CAAM job ring may be used by a different process or guest operating system or by TrustZone SecureWorld, CAAM enforces access control independently for each job ring. This is done by comparing the owner of the job ring with the access group membership of the secure memory partition that is being accessed.

### 5.8.6.4.2.3 Setting access control permissions

Two registers are used to specify access permissions for a partition. One of these registers is the Secure Memory Access Groups (SMAG) Register, which specifies which of the bus master identities can access the memory assigned to this partition:

- If a bus master (including CAAM) accesses a partition using a MID that is not listed in one of the partition's two SMAGJR registers, the bus master is denied access, regardless of the access permissions specified in the partition's other access permissions register, the Secure Memory Access Permissions (SMAPJR) Register.
- If the access is permitted by the SMAG registers, the SMAPJR register is consulted to see if the attempted access type (for example, read, write, blob operation) is allowed for that access group.
- If the access is denied on the basis of access group membership or access type, secure memory returns a bus error.
- If the access was attempted by a job descriptor, the execution of the job descriptor terminates with a bus error code.

A partition is claimed and the initial access permissions for the partition are established by writing into the partition's SMAPJR register by means of one of the job ring register pages. .

The appropriate JRMID register is determined by the SMAPJR register address that was written. For example, if the SMAPJR register address is within the job ring 2 4 KB register address space, JR2 MID is selected. If there is a match (and the partition was

unallocated), the controller marks the partition as allocated to the owner of job ring 2. Thereafter, only bus transactions using that particular MID can write into the partition's SMAPJR and SMAG2/1JR registers. The access control information written into the SMAPJR Register indicates which types of bus transactions are permitted for that partition. The transaction types that may be specified in the SMAPJR Register are as follows:

- Read
- Write
- Secure memory blob export/import permitted (even if read or write is prohibited)

The SMAPJR Register also includes some other information about the partition, such as a key modifier field (see [Blob encapsulation](#)), lock bits for the SMAPJR and SMAG2/1JR registers, whether it is a critical security parameter (CSP), and whether it is a public security parameter (PSP) partition.

#### NOTE

CSP partitions are intended to hold data whose confidentiality must be protected. Therefore, as discussed in [Automatic RAM zeroization](#), pages assigned to CSP partitions are zeroized if the partition or the pages are de-allocated or upon detection of tampering or other security violations.

A secure memory partition may contain software or data whose integrity and availability are security-critical. For example, the partition may contain software that is intended to be invoked when a security violation occurs. The partition and its individual pages should not be de-allocated. Setting the PSP flag in a partition's SMAPJR Register prevents the partition from being de-allocated, and prevents any pages from being de-allocated from that partition. However, if a PSP partition is also marked as CSP, the partition's pages will still be zeroized in the event of a security violation.

The SMAPJR Register also contains two lock bits, one that applies to the SMAG2JR register and SMAG1JR register and one that applies to most fields of the SMAPJR register. Once the SMAG\_LCK bit has been set, no further changes can be made to any bits in the SMAG2/1JR registers. Once the SMAP\_LCK bit has been set, the only fields in the SMAPJR register that can be changed are the SMAG\_LCK bit, which can be changed to a 1 if it is currently a 0, and the PARTITION\_KMOD field, which can still be changed to any value at any time. After a lock bit is set, it cannot be reset until the partition is de-allocated, or the chip undergoes a power-on reset.

### 5.8.6.4.3 Cryptographic protection of exported data

CAAM implements a special type of blob (see [Blob types differentiated by memory type](#)) to enable data to be safely exported from, and later restored to, a secure memory partition.

#### 5.8.6.4.3.1 Exporting/importing memory type blobs

When used with secure memory, general memory blobs are subject to the G1/2 read and write permissions in the SMAPJR registers of the accessed partitions, but are not subject to the blob permission. Secure memory blobs are subject only to the G1/2 BLOB permissions bits. If the G1\_BLOB bit is 1, a descriptor executing with an ACCESS\_GROUP1 MID can export a secure memory blob even if the G1\_READ bit is 0, and can import a secure memory blob even if the G1\_WRITE bit is 0. The same is true for the G2 bits. This allows data from partitions to be backed up to nonvolatile storage, or swapped in and out of secure memory for other reasons, even if read permission or write permission is not set. See [Blob types differentiated by memory type](#) for more information.

#### 5.8.6.4.3.2 Access permissions cryptographically bound to secure memory blobs

The BKEK for secure memory blobs is created in a manner that cryptographically binds the access permissions of the secure memory partition to the blob. As a consequence, any attempt to decapsulate a secure memory blob into a partition with different access permissions fails because the BKEK would be derived incorrectly, which leads to the BK also being decrypted incorrectly. This then causes the data blob to be decrypted incorrectly and the MAC tag check to fail. Although the destination addresses would be overwritten, the decapsulated data would be indecipherable. Note that in this case the decapsulation process would terminate with an error indication.

A secure memory blob does not have to be decapsulated into the same partition from which it was exported, as long as the two partitions have identical access permissions and are owned by the same job ring.

### 5.8.6.5 Initializing secure memory

At POR, the secure memory automatically initializes itself by overwriting all pages with a known data pattern, allocating all pages to partition 0, and assigning the ownership of partition 0 to job ring 0.

The SMAPJR and SMAG2/1JR registers for partition 0 are set to allow any type of access by any master identifier. Thus, by default, secure memory appears as ordinary non-access controlled memory, and can be used for that purpose without software initialization. However, if software wishes to use the access control features of secure memory, further initialization is required to specify the access permission settings.

If one partition is sufficient to implement the desired access control policy, then all that is required is for the owner of job ring 0 to write the appropriate values to SMAPJR0 and/or SMAG2/1JR0. Because the JRMID register resets to all 0s at POR, by default the owner of partition 0 is MID 0. However, this value can be changed by the manager processor (which is specified by the MID on the caam\_ips\_manager[2:0] input signals) or TrustZone SecureWorld.

If the desired access control policy requires more than one set of access privileges, then additional secure memory partitions must be allocated pages and given appropriate access control settings. Because all pages are allocated to partition 0 at POR, some of these pages must be reallocated to other partitions. To do this partition 0's owner or TrustZone SecureWorld must either de-allocate one or more pages from partition 0, or de-allocate the entire partition, which frees all of its pages. De-allocating pages or partitions, or allocating pages to partitions, is accomplished via the Secure Memory Command Register. Any job ring owner or TrustZone SecureWorld can claim an unclaimed partition by writing access control settings into the partition's SMAPJR register. The partition's new owner can then write to the partition's SMAG2/1JR registers to limit access to the partition to certain bus master IDs .



# **Chapter 6**

## **Secure Non-Volatile Storage (SNVS)**

### **6.1 SNVS overview**

Secure Non-Volatile Storage (SNVS) is a companion logic block to the Cryptographic Acceleration and Assurance Module (CAAM). This block is the chip's central reporting point for security-relevant events, such as the success or failure of boot software validation and the detection of potential security compromises. This security event information allows the SNVS to determine whether the chip is in a trustworthy state.

If the SNVS determines the chip is in a trustworthy state, it allows CAAM to use a secret 256-bit value that CAAM uses to derive cryptographic keys during blob encapsulation and decapsulation. The secret value is either a one-time programmable master key (OTPMK) stored in fuses, a zeroizable master key (ZMK) stored in the low-power section, or a combination of the two. Because the ZMK value is cleared upon security violation, selecting the ZMK value as a component of the secret value prevents blobs from being decapsulated even on subsequent boot cycles that result in a trustworthy state. The ZMK can be programmed either by software or directly by hardware through a dedicated interface to CAAM's random data generator (RNG) block. Blobs are encrypted data structures that can be used to protect sensitive data such as private keys, DRM keys, and proprietary software.

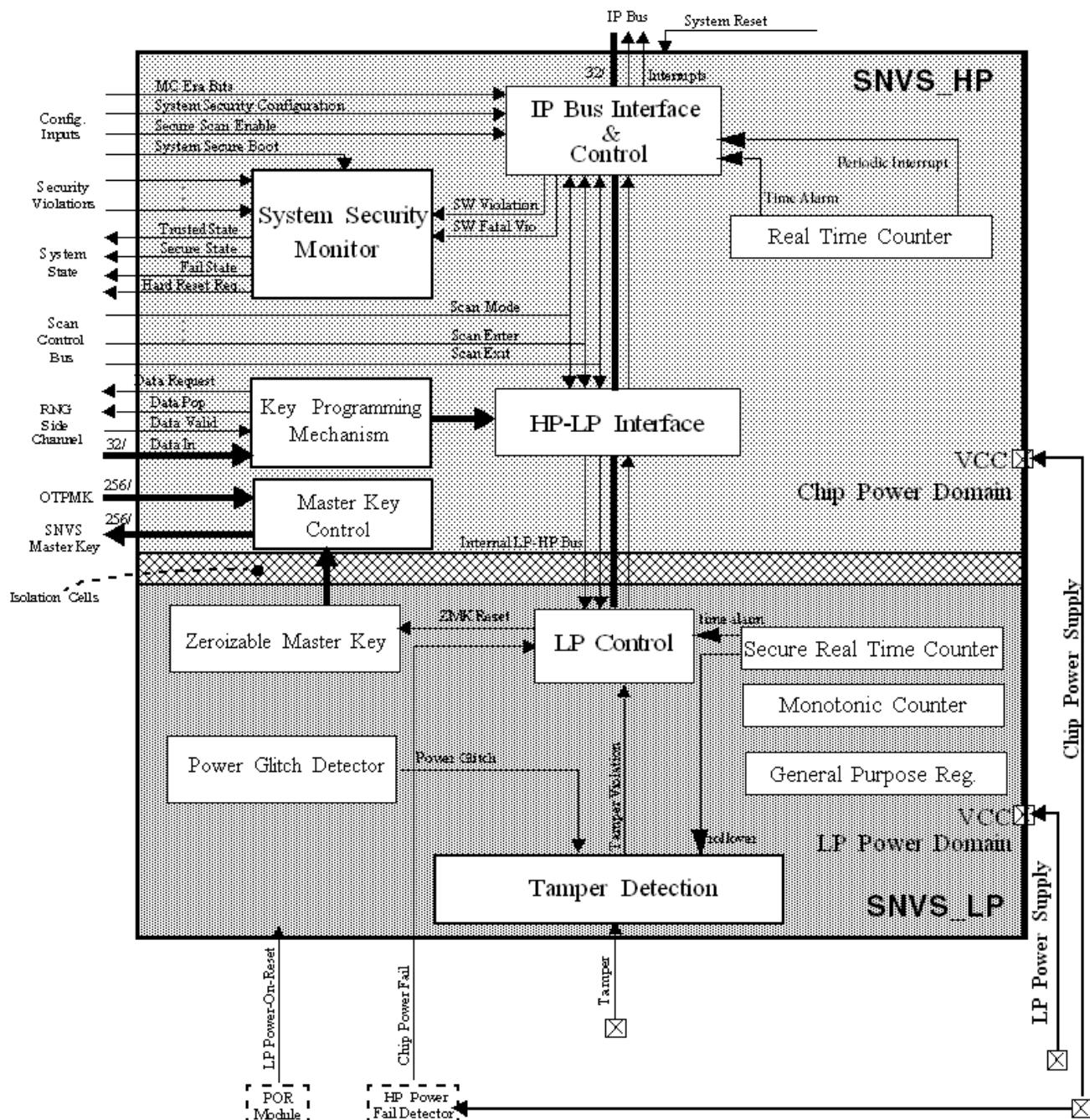
When the chip is in the non-secure state, CAAM cannot decapsulate blobs that were encapsulated while CAAM was in a trustworthy state (either secure state or trusted state).

In addition to acting as a central reporting point for system security violations, this block provides secure non-volatile data storage by means of battery-backed internal registers. The low-power (battery-backed) section incorporates a secure real time counter, a monotonic counter, and a general purpose register.

#### **6.1.1 SNVS feature summary**

The following figure presents a top-level diagram of the SNVS block.

## SNVS overview



**Figure 6-1. SNVS block diagram**

The following table summarizes the features:

**Table 6-1. SNVS feature summary**

Feature	What it does
System security monitor (SSM)	<ul style="list-style-type: none"> <li>• Receives configuration inputs from the security fuse processor</li> <li>• Receives security violation inputs from the various detectors in the chip</li> <li>• Tracks security state</li> <li>• Generates security state outputs to the CAAM and other logic within the chip</li> <li>• Can request system hard reset in case of non-recoverable violation</li> <li>• Programmable high assurance counter (HAC) to control time delay before system hard reset request generation</li> </ul>
Master key control	<ul style="list-style-type: none"> <li>• Performs validity checks for the one time programmable master key and zeroizable master key before allowing the CAAM to use them</li> <li>• Selects between OTPMK, ZMK, and a test master key output according to the system condition and configuration</li> </ul>
Zeroizable master key (ZMK)	<ul style="list-style-type: none"> <li>• The ZMK state is nonvolatile</li> <li>• The ZMK can be programmed either by software or directly by hardware</li> <li>• Interface to the random number generator for direct hardware programming</li> <li>• The value of the key can be selected for use by hardware cryptographic accelerators</li> <li>• The key value is zeroized in case of security violation</li> </ul>
Secure real time counter (SRTC)	<ul style="list-style-type: none"> <li>• The SRTC state is nonvolatile</li> <li>• The counter is driven by a dedicated clock, which should always be functional independent of the chip configuration and main chip power state.</li> <li>• The counter is a non-rollover counter</li> <li>• Programmable time alarm interrupt. <ul style="list-style-type: none"> <li>• This alarm generates an interrupt to alert the processor when the time alarm is enabled and the system is powered up.</li> <li>• This alarm generates a wake-up alarm via an external pin when the time alarm is enabled, the wake-up external alarm is enabled, and the system is powered down.</li> </ul> </li> <li>• The time value is invalidated in case of a security violation.</li> </ul>
Non-secure Real Time Counter	<ul style="list-style-type: none"> <li>• The counter is driven by a dedicated clock, which is off when the system power is down.</li> <li>• The counter can be synchronized to the value of the Secure Real Time Counter.</li> <li>• Programmable time alarm interrupt</li> <li>• Periodic interrupt can be generated with different frequencies.</li> </ul>
Real time counter (RTC)	<ul style="list-style-type: none"> <li>• The counter is driven by a dedicated clock, which is off when the system power is down</li> <li>• The counter can be synchronized to the value of the Secure Real Time Counter</li> <li>• Programmable time alarm interrupt</li> <li>• Periodic interrupt can be generated with different frequencies</li> </ul>
Monotonic counter	<ul style="list-style-type: none"> <li>• The monotonic counter state is nonvolatile.</li> <li>• The counter can only increment.</li> <li>• The counter is a non-rollover counter</li> <li>• The counter value is invalidated in case of security violation.</li> </ul>
General-purpose register	<ul style="list-style-type: none"> <li>• The general-purpose register state is nonvolatile.</li> </ul>
Register access protection	<ul style="list-style-type: none"> <li>• Privileged software access policy</li> <li>• Registers can be programmed only when the system security monitor is in functional state.</li> <li>• Some registers/values can only be programmable once per boot cycle.</li> </ul>

*Table continues on the next page...*

**Table 6-1. SNVS feature summary (continued)**

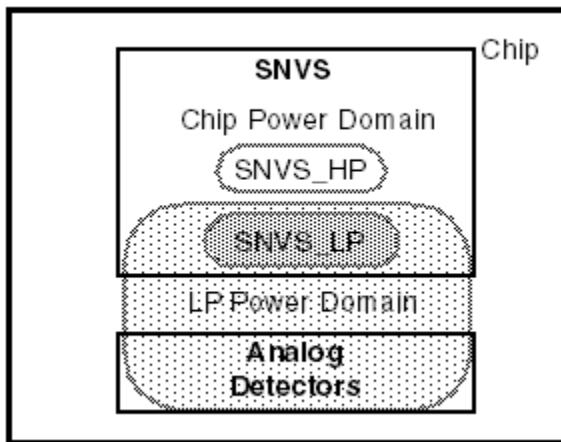
Feature	What it does
Violation/tamper detection and reporting	<ul style="list-style-type: none"> <li>• Detects (internal to the block) the following security violations:           <ul style="list-style-type: none"> <li>• Scan exit</li> <li>• Power glitch</li> <li>• Invalid OTPMK (ECC check failure)</li> <li>• ZMK ECC check failure</li> <li>• SRTC rollover</li> <li>• MC rollover</li> </ul> </li> <li>• Detects (receives from detectors external to the block) the following security violations:           <ul style="list-style-type: none"> <li>• Software violations</li> <li>• Six security violation inputs</li> <li>• External tamper input</li> <li>• Second external tamper input</li> <li>• Clock tamper input</li> <li>• Voltage tamper input</li> <li>• Temperature tamper input</li> <li>• Two wire-mesh tamper input</li> </ul> </li> <li>• External tamper detection logic can be connected to a security violation input</li> <li>• Direct connections to CAAM to lock out access to the OTPMK/ZMK, force zeroization of sensitive information</li> <li>• Configurably triggers a device hard reset</li> <li>• Reports to software (interrupts) all security violation and functional events</li> </ul>

## 6.2 SNVS structure

The block is divided into two major submodules based on power supply: the high power part (SNVS\_HP) and the low power part (SNVS\_LP). They are powered as follows:

- SNVS\_LP- dedicated always-powered-on domain
- SNVS\_HP - system (chip) power domain

The following figure illustrates the low power and chip power domains.



**Figure 6-2. SNVS power supply**

SNVS \_HP implements all features that enable system communication and provisioning of the SNVS \_LP module. It also incorporates the system security monitor, which controls the system security state, and the master key control block, which is responsible for checking and selecting the master key value.

SNVS \_LP provides hardware that enables secure storage and protection of sensitive data. It detects and responds to the real time security threats even when system is in power-down mode.

### 6.3 SNVS \_HP (high power part)

SNVS \_HP is partitioned into the following functional units:

- IP bus interface
- SNVS \_LP interface
- System security monitor
- Master key control block
- Zeroizable master key programming mechanism
- Real time counter with alarm
- Control and status registers

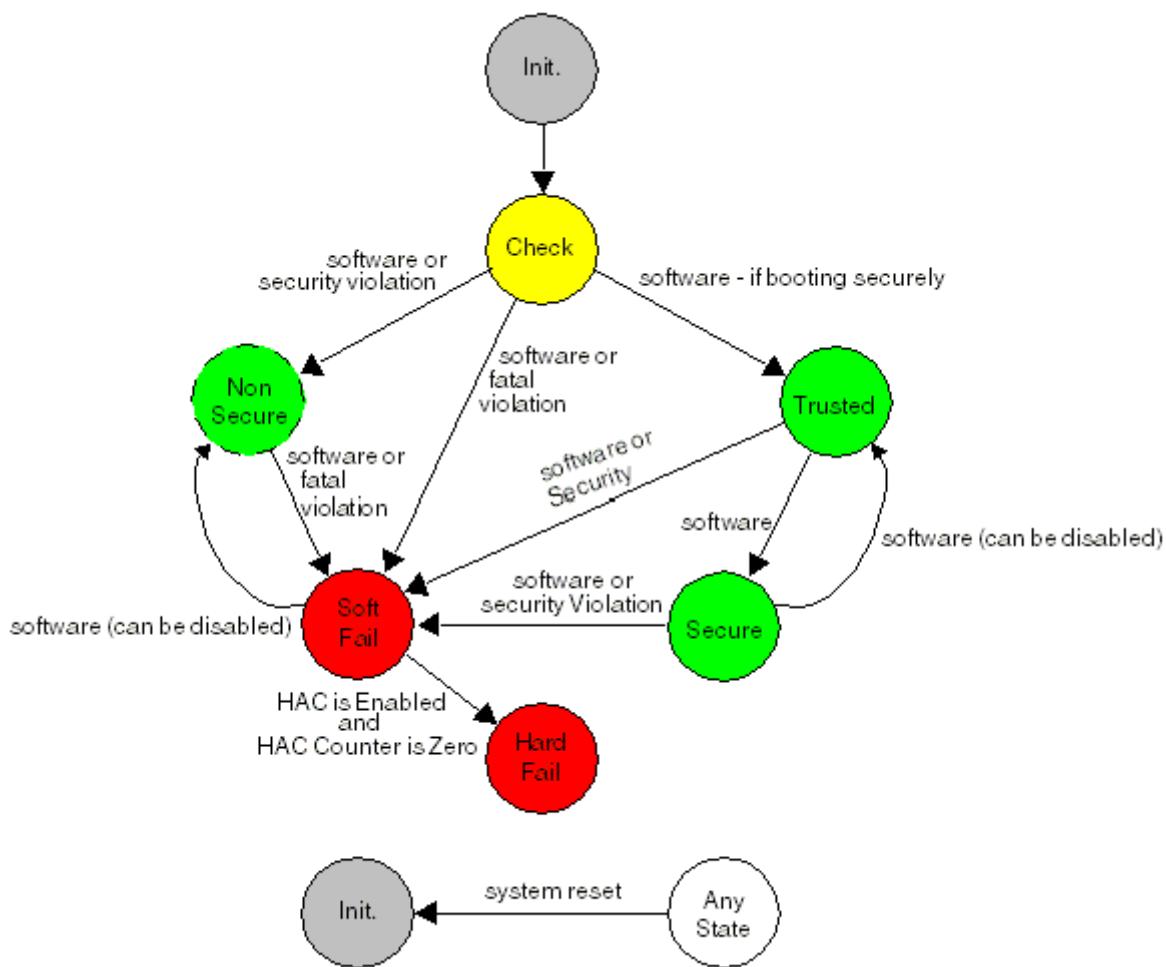
SNVS \_HP is in the chip's power supply domain and thus receives power along with the rest of the chip. SNVS \_HP provides an interface between SNVS \_LP and the rest of the system; there is no way to access the SNVS \_LP registers except through the SNVS \_HP. For access to the SNVS \_LP registers, SNVS \_HP must be powered up. It uses a register access permission policy to determine whether access to particular registers is permitted.

### 6.3.1 System security monitor (SSM)

The system security monitor (SSM) monitors system security violations and controls the security state of the system. In essence, the SSM is a state machine. It classifies security violation sources as disabled, fatal, or non-fatal, and each classification results in different state transitions. A security violation source that is classified as a fatal security violation always results in the SSM transition to the soft fail state. A non-fatal security violation results in a transition from check to non-secure or from trusted/secure to soft fail states. The SSM does not react to a disabled violation. See [HP security violation policy](#) for a detailed description of the SNVS security violation policy.

#### 6.3.1.1 Transitioning among system security monitor states

The following figure and tables explain the SSM's states and transitions.



**Figure 6-3. SSM state machine**

**Table 6-2. State definitions**

<b>State</b>	<b>What happens</b>	<b>State transition</b>
Init	System enters this state on the system POR. In this state, the SSM ignores all security violations sources; security violations are not recorded and not responded to in the SNVS _HP domain. SNVS registers cannot be programmed in this state.	SSM transitions to the check state .
Check	System performs the check sequence. SNVS registers cannot be programmed in this state, with the exception of several bits in the HP Command Register.	<ul style="list-style-type: none"> <li>Software sets the control bit in the HP Command Register to initiate the transition from check to Trusted State. The transition is allowed if the system is booting internally or if System Security Configuration is Fab (000) . Upon transition from check state to trusted state, SNVS _LP is reset if it was previously provisioned in the non-secure state.</li> <li>If the SSM detects a non-fatal security violation condition while in this state, it immediately transitions to the non-secure state.</li> <li>If the SSM detects a fatal violation, it transitions to the soft fail state if fatal violation is detected.</li> </ul>
Non-Secure	In this state, the secure and trusted mode indication signals are not set. In non-secure state any write accesses to the SNVS _LP registers are denied if SNVS _LP was provisioned in the trusted/secure state.	SSM transitions to the soft fail state if a fatal violation is detected.
Trusted	In this state, the trusted and secure mode indication signals are asserted.	<p>Privileged software triggers the transition from secure state to trusted state or vice versa by writing to the SSM_ST bit in the HP Command Register. The secure state to trusted state transition can be disabled by setting the SSM_ST_DIS bit in the HP Command Register.</p> <p>If the SSM detects a security violation condition, it immediately (without clock) transitions to the soft fail state.</p>
Secure	In this state, the secure mode indication signal is asserted and trusted mode indication signal is de-asserted.	<p>Privileged software triggers the transition from trusted state to secure state or vice versa by writing to the SSM_ST bit in the HP Command Register. The secure state to trusted state transition can be disabled by setting the SSM_ST_DIS bit in the HP Command Register.</p> <p>If the SSM detects a security violation condition, it immediately (without clock) transitions to the soft fail state.</p>

*Table continues on the next page...*

**Table 6-2. State definitions (continued)**

State	What happens	State transition
Soft fail	In this state SSM generates a security interrupt and the fail state indication output, which is treated as a security violation by other security modules. SNVS registers cannot be programmed in this state, except for several bits in the HP Command Register.	If high assurance configuration has been enabled by setting the bit in the HP Command Register, the SSM transitions to the hard fail state after the HAC counter counts down to zero. Software can stop this counter to prevent the transition to the hard fail state.  Software triggers the transition to the non-secure state only if the HAC counter is deactivated (disabled or stopped) and the transition to the non-secure state is not disabled. Note that in cases of active fatal security violation, the SSM stays in the soft fail state.
Hard Fail	Entering this state triggers hard reset request output, which should be used in the system to perform a hardware reset without the aid of software. SNVS registers cannot be programmed in this state.	The SSM remains in the hard fail state until the system is reset.

**Table 6-3. SSM state transition table**

Next State (transition to next state determined by conditions listed in the table cells below)							
Current State	Init	Check	NonSecure	Soft Fail	Hard Fail	Trusted	Secure
Init	SII other conditions	LP Section is Powered On	—	—	—	—	—
Check	System Reset	All other conditions	Non-Fatal Security Violation or write 1 to the SSM_ST bit of the HPCOMR when non-secure boot	Fatal Security Violation <sup>1</sup>	—	Write 1 to the SSM_ST bit of the HPCOMR when secure boot	—
NonSecure	System Reset	—	All other conditions	Fatal Security Violation <sup>1</sup>	—	—	—
Soft Fail	System Reset	—	Write 1 to the SSM_ST bit of the HPCOMR (if not disabled by SSM_SFNS_DIS bit of the HPCOMR and HAC Counter4 is either disabled or stopped)	All other conditions	HAC is Enabled & HAC Counter is Zero	—	—
Hard Fail	System Reset	—	—	—	All other conditions	—	—
Trusted	System Reset	—	—	Non-Fatal or Fatal Security Violation <sup>1</sup>	—	All other conditions	Write 1 to the SSM_ST bit of the HPCOMR

Table continues on the next page...

**Table 6-3. SSM state transition table (continued)**

Current State	Next State (transition to next state determined by conditions listed in the table cells below)						
	Init	Check	NonSecure	Soft Fail	Hard Fail	Trusted	Secure
Secure	System Reset	—	—	Non-Fatal or Fatal Security Violation <sup>1</sup>	—	Write 1 to the SSM_ST bit of HPCOMR (if not disabled by SSM_ST_DIS bit of the HPCOMR)	All other conditions

1. See [Table 1](#) for a list of non-fatal and fatal security violations

### 6.3.1.2 HP security violation policy

All HP security violation sources are classified into three categories: disabled, non-fatal security, and fatal security.

- Disabled violation-SSM does not react on this violation.
- Non-fatal security violation-Results in the following transitions:
  - Check state → non-secure state
  - Trusted state → soft fail state
  - Secure state → soft fail state
- Fatal security violation-SSM transitions to the soft fail state from check, non-secure, trusted, or secure states

Regardless of the category all security violation events are recorded in the corresponding status registers.

### 6.3.1.3 SNVS \_HP violation sources

The following table explains all SNVS \_HP security violation sources.

**Table 6-4. SNVS \_HP security violation sources**

Security violation source	Default behavior	Configuration options	Comments
Scan exit violation	Fatal	-	Asserted upon scan exit and stays active till system reset
Software fatal violation	Fatal	-	Asserted by software
Software non-fatal violation	Non-Fatal	-	Asserted by software

*Table continues on the next page...*

**Table 6-4. SNVS \_HP security violation sources  
(continued)**

Security violation source	Default behavior	Configuration options	Comments
Bad master key violation	Non-Fatal	-	Asserted when: - OTPMK does not pass validity check - ZMK ECC Check failure (if ZMK ECC Check is enabled) - ZMK is zero when it is selected for use
Security violation Inputs 0-4	Non-Fatal	• Non-fatal • Fatal	Asserted on the security violation input ports 0-4
Security violation input 5	Disabled	• Disabled • Non-fatal • Fatal	Asserted on the security violation input port 5
From LP section			
LP power-on-reset	Fatal	-	Asserted upon LP POR detection. This signal is asynchronous to any clock in the module.
LP security violation	Disabled	• Disabled • Non-Fatal • Fatal	Asserted in the LP section. This signal can be asserted asynchronously to any clock in the module.

### 6.3.2 Master key control

The master key control block functionality depends on the particular security conditions and configuration. The master key control block selects among the following master key sources:

- One time programmable master key (OTPMK)

The OTPMK is a random value stored in the non-volatile memory (typically fuses) outside the SNVS module.

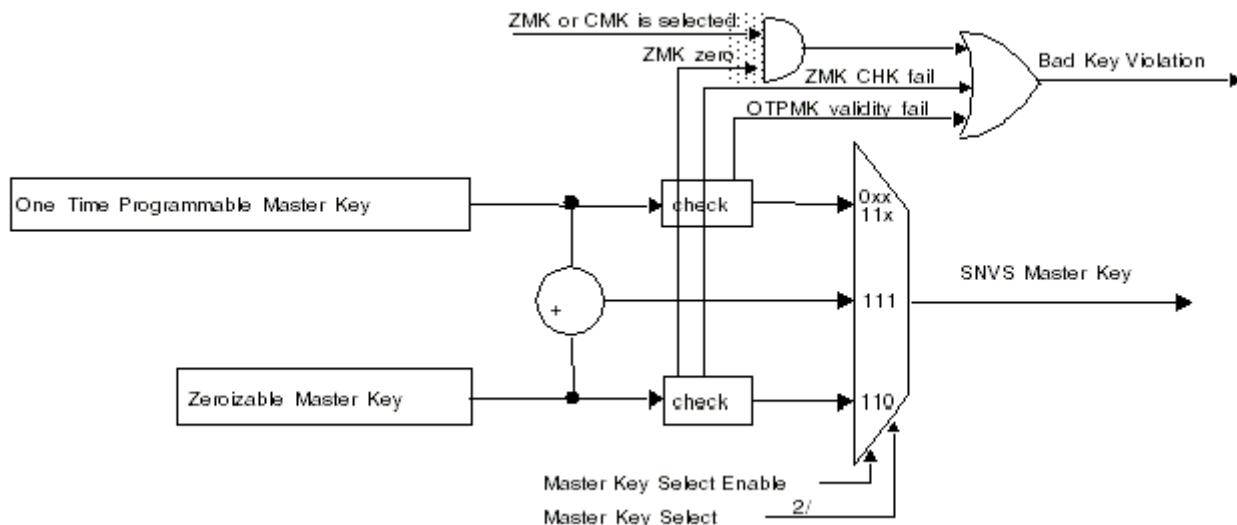
- Zeroizable master key (ZMK)

The ZMK is a random value stored in the ZMK registers of the SNVS \_LP section.

- Combined master key (CMK)

This key is created by a bitwise XOR operation of OTPMK and ZMK values.

The following figure shows the master key selection scheme.



**Figure 6-4. Master key control scheme**

The master key control block incorporates hardware checking logic for OTPMK and ZMK values to assure that the keys are programmed with their intended valid values and have not been corrupted.

The OTPMK's value is checked for whether it is all zeros. The OTPMK can be optionally checked for whether it has generated a valid nine-bit Hamming codeword (see [Error Code for the OTPMK](#), for more information). If the OTPMK is found to be all zeros or to be an invalid codeword, the validity check fails.

If ZMK is selected for use and marked as valid, it is also checked for whether it different from the default all-zero value. In addition, the ZMK can be optionally checked for whether it has generated a valid nine-bit Hamming codeword.

If the validity check fails, the master key control block reports the bad key violation to the system security monitor, which treats it as a non-fatal violation. The check results can always be found in the HP status registers.

### 6.3.2.1 Error Code for the OTPMK

The Hamming code used for the 256-bit OTPMK has nine code bits. It can detect all 1, 2, and 3 bit errors in the codeword. It also detects most, but not all, errors of more than 3 bits. Therefore, the OTPMK actually contains 247 random bits.

Numbering the bits in the OTPMK from zero to 255, the code bits are bits 0, 1, 2, 4, 8, 16, 32, 64, and 128. All remaining bits are data bits. Each of these code bits is the XOR of a subset of the bits in the entire code word.

To determine which bits are used to form each code bit, look at the binary representation of the bit position of each code bit (ignoring bit zero for the time being) in the following table.

**Table 6-5. Error codes**

Code bit	Binary representation
0	00000000
1	00000001
2	00000010
4	00000100
8	00001000
16	00010000
32	00100000
64	01000000
128	10000000

For code bit number 1, there is a single one in its binary representation. This code bit is the XOR of all bit positions that also have a one in this same point in their binary representation (i.e., all odd bit positions). The same is true for the other code bits, excepting code bit 0. For example, code bit 2 is the XOR of bits 3, 6, 7, 10, 11, 14, 15,..., 254, 255, and code bit 128 is the XOR of all bits from 129 through 255 inclusive. After all of the other code bits have been calculated, code bit zero is simply the XOR of all of the other bits, including the code bits.

Another way of looking at this is to ask which code bits does a data bit affect. If we look at the binary representation of the bit position of a data bit, the 1s represent the code bits that this data bit affects. For example, data bit 99 (01100011) is XORed into code bits 1, 2, 32, 64, and bit 0 (the overall parity bit).

### 6.3.2.2 Generating the code bits

To generate a code word to be programmed into the one time programmable elements for the OTPMK, it is simplest to start with an array of 256 random bits (the array numbered 0 through 255). The following pseudocode then replaces the nine code bits with their proper values. The starting random bits at those locations are discarded.

```
// Generate the Hamming code bits for the 256 bits
// stored in Number. The values at the locations of the code bits
// are ignored and is overwritten with the generated values.
Generate_code_bits(bool Number[256])
{
    int i, j;

    // Calculate each code bit in turn
    for(i = 1; i <= 128; i = (i << 1)) {
```

```

// Examine each data bit
// Only bits greater than i need to be checked as no
// bit less than i will ever be XOR'ed into i
// J starts at i so that Number[i] is initialized to 0
for(j = i; j <= 255; j = j + 1) {
    if ( (i & j) != 0) {
        Number[i] = Number[i] ^ Number[j];
    }
}
// Calculate the overall parity
// J starts at 0 so that Number[0] is initialized to 0
// Number[0] contains the even parity of all of the bits
for(j = 0; j <= 255; j = j + 1) {
    Number[0] = Number[0] ^ Number[j];
}

```

### 6.3.2.3 Checking the code bits

The logic within the SNVS uses an equivalent of the following pseudocode to check whether the 256-bit OTPMK value is a legal codeword or if any of the bits were not programmed properly.

```

// Check the Hamming code bits for the 256 bits stored in Number.
// If an error is detected, return BAD.
// If no errors are detected, return GOOD.
bool Check_code_bits(bool Number[256])
{
    int i, j, n;
    bool Check[9];
    // Calculate each code bit in turn
    // i steps through Number
    // n steps through Check; n = log(i)
    for(i = 1, n=0; i <= 128; i = (i << 1), n = n + 1) {
        // Examine each data bit
        // Only bits greater than i need to be checked as no
        // bit less than i will ever be XOR'ed into Check[n]
        Check[n] = 0;
        for(j = i; j <= 255; j = j + 1) {
            if ( (i & j) != 0) {
                Check[n] = Check[n] ^ Number[j];
            }
        }
    }
    // Calculate the overall parity
    // Check[8] is the even parity of all of the bits
    Check[8] = 0;
    for(j = 0; j <= 255; j = j + 1) {
        Check[8] = Check[8] ^ Number[j];
    }
    // Any non-zero Check bit means an error has been detected
    for(n = 0; n <=8; n = n + 1) {
        if(Check[n] != 0) {
            return BAD;
        }
    }
    return GOOD;
}

```

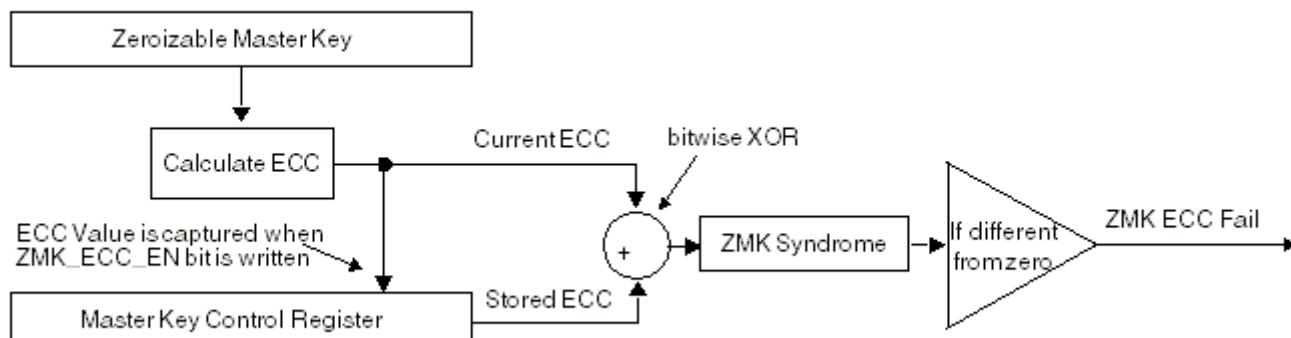
### 6.3.2.4 Error code for the ZMK

The Hamming code used for the 256-bit ZMK has nine code bits. It can detect all 1, 2 and 3 bit errors in the codeword. It also detects most, but not all, errors in more than 3 bits. Unlike OTPMK, where the check code bits are part of the OTPMK value, the ZMK code bits are stored separately from the ZMK value. Therefore, the presence of code bits does not reduce the randomness of the ZMK value.

SNVS logic automatically calculates and stores the ZMK ECC value when software enables the ZMK ECC feature. Software is not allowed to write this value. If read access to the ZMK registers is not allowed, software cannot read the ZMK ECC value.

Once the ZMK ECC checking mechanism is enabled and the ZMK ECC value is captured, the SNVS logic continuously checks the stored ECC value against the current ECC value. If the current ECC value is found to be different than the one stored in the register, the ZMK ECC failure violation is generated and the ZMK syndrome value is captured.

The following figure shows the flow:



**Figure 6-5. ZMK error code programming and checking flow**

The steps are as follows:

1. Use either software or the hardware programming mechanism to program the ZMK value.
2. Write the ZMK\_ECC\_EN bit of the SNVS \_LP Master Key Control Register (LPMKCR) to enable the ZMK ECC mechanism.
3. Hardware calculates ZMK\_ECC\_VALUE's value and sets it automatically.
4. SNVS logic checks whether ZMK\_ECC\_VALUE is different from the current ZMK ECC value.
5. If it is different, the ZMK\_ECC\_FAIL and ZMK\_SYNDROME fields of the SNVS \_HP Security Violation Status Register (HPSVSR) are set and a security violation is generated.

### 6.3.3 ZMK hardware programming mechanism

The ZMK\_HWP bit controls the ZMK programming flow. When ZMK\_HWP is set, only hardware can program the ZMK. Otherwise, only software can program the ZMK. In either case, the ZMK cannot be programmed if it is locked for writes.

To initiate hardware programming of the ZMK, software sets the PROG\_ZMK bit of the HP Command Register. In hardware programming mode, the ZMK\_VAL bit is set automatically after the ZMK's last word is programmed. This bit (ZMK\_VAL) indicates to the software whether the programming is still in progress or has already completed.

### 6.3.4 Non-secure real time counter

SNVS \_HP has an autonomous non-secure real time counter. The counter is not active and is reset when the system is powered down. The HP RTC can be used by any application; it has no privileged software access restrictions. The counter can be synchronized with the SNVS\_LP SRTC by writing to a specific bit in the SNVS\_HP Control Register.

#### 6.3.4.1 Calibrating the time counter

The RTC accuracy may suffer from a drift in the clock, which is used to increment the RTC register. To compensate for this drift, a clock calibration mechanism can adjust the RTC value. It is up to the system processor to decide whether calibration is required or not. If RTC correction is required, enable the mechanism and set the calibration value in the control register. The calibration value is a 5 bit value including the sign bit, which is implemented in 2's complement.

If the calibration mechanism is enabled, the calibration value is added or subtracted from the RTC on a periodic basis, once per 32768 cycles of the RTC clock. The following table shows the available correction range.

**Table 6-6. Time counter calibration settings**

Calibration value setting	Correction in counts per 32768 cycles of the counter clock
01111	+15
:	:
00010	+2
00001	+1
00000	0

*Table continues on the next page...*

**Table 6-6. Time counter calibration settings (continued)**

Calibration value setting	Correction in counts per 32768 cycles of the counter clock
11111	-1
11110	-2
:	:
10001	-15
10000	-16

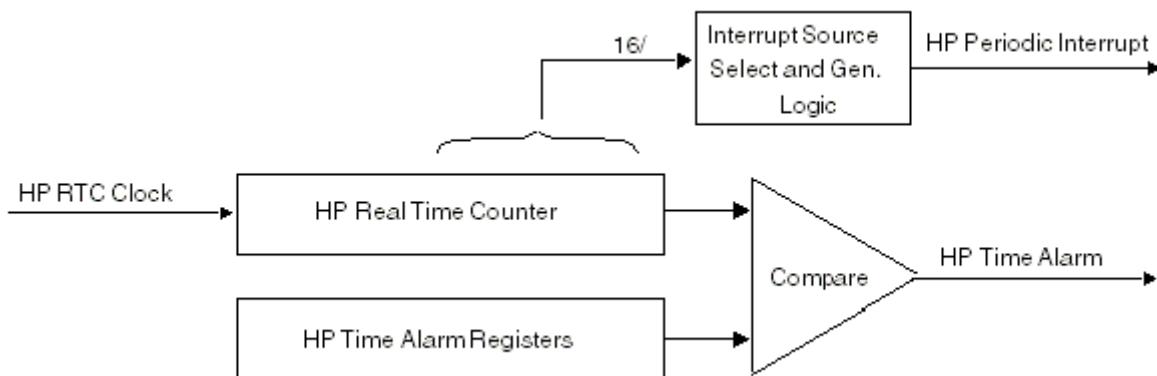
### 6.3.4.2 Time counter alarm

The SNVS \_HP non-secure RTC has its own time alarm register. Any application can update this register. The SNVS \_HP time alarm can generate interrupts to alert the host processor and can wake up the host processor from one of its low-power modes (wait, doze, stop). Note that this alarm cannot wake up the entire system if it is powered off because this alarm would also be powered off.

### 6.3.4.3 Periodic interrupt

The SNVS \_HP non-secure RTC incorporates a periodic interrupt. The periodic interrupt is generated when a zero-to-one or one-to-zero transition occurs on the selected bit of the RTC. The periodic interrupt source is chosen from 16 bits of the HP RTC according to the PI\_FREQ field setting in the HP Control Register. This bit selection also defines the frequency of the periodic interrupt.

The following figure shows the SNVS \_HP RTC and its interrupts.

**Figure 6-6. SNVS \_HP RTC, alarm, and interrupts**

## 6.4 Low power part (SNVS \_LP)

SNVS \_LP has the following functional units:

- Zeroizable master key
- Secure non-rollover real time counter with alarm
- Non-rollover monotonic counter
- Power glitch detector
- General purpose register
- Control and status registers

It also incorporates security violation and tamper detection logic.

The SNVS \_LP is a data storage subsystem with enhanced security capabilities. Its purpose is to store and protect system secure data, regardless of the main system power state.

SNVS \_LP is in the always-powered-up domain, which is a separate power domain with its own power supply.

### 6.4.1 Behavior during system power down

When the chip power supply domain loses power, SNVS \_LP continues to operate normally, and it ignores all inputs from SNVS \_HP. The tamper input signal remains functional

### 6.4.2 Zeroizable master key (ZMK)

SNVS \_LP incorporates logic for storage of a 256-bit ZMK value. This value (or the ZMK XORed with the OTPMK) can be selected as the master key input to CAAM .

Either hardware or software can program the ZMK value. In hardware programming mode, software cannot read the ZMK value. In software programming mode, software can read the ZMK value before it is locked.

When there is a security violation, the ZMK is asynchronously zeroized and invalidated.

### 6.4.3 Secure real time counter (SRTC)

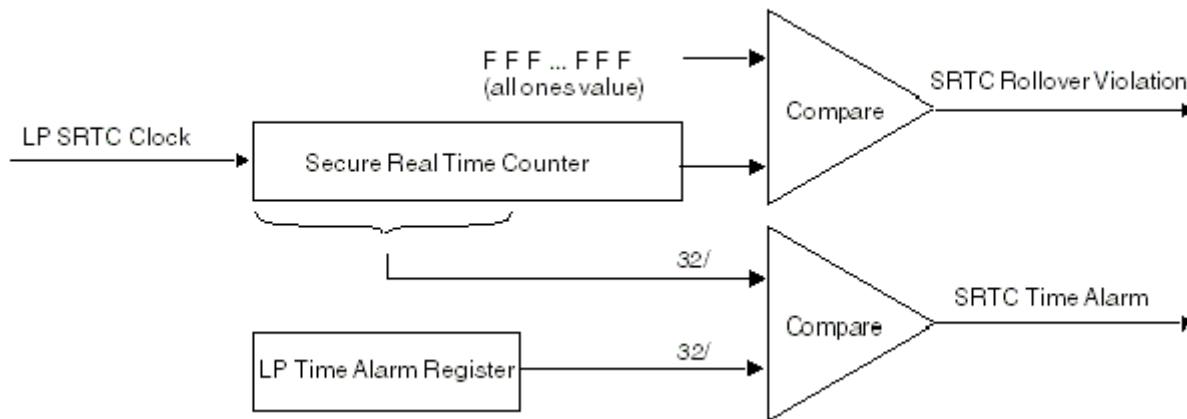
The SNVS\_LP incorporates an autonomous SRTC. This is a non-rollover counter, which means that the SRTC does not rollover when it reaches the maximum value of all ones. Instead, a time rollover indication is generated to the SNVS\_LP tamper monitor, which generates a security violation and interrupt.

#### 6.4.3.1 Calibrating the SRTC time counter

To compensate for possible drift in the SRTC clock source, use the clock calibration mechanism implemented with the SRTC. This mechanism works the same way as the non-secure RTC clock calibration mechanism. Refer to the [Calibrating the time counter](#) for the detailed description of its functionality.

#### 6.4.3.2 Time counter alarm (zmk)

The following figure shows the SNVS\_LP Secure Real Time Counter, time alarms, and rollover security violation.

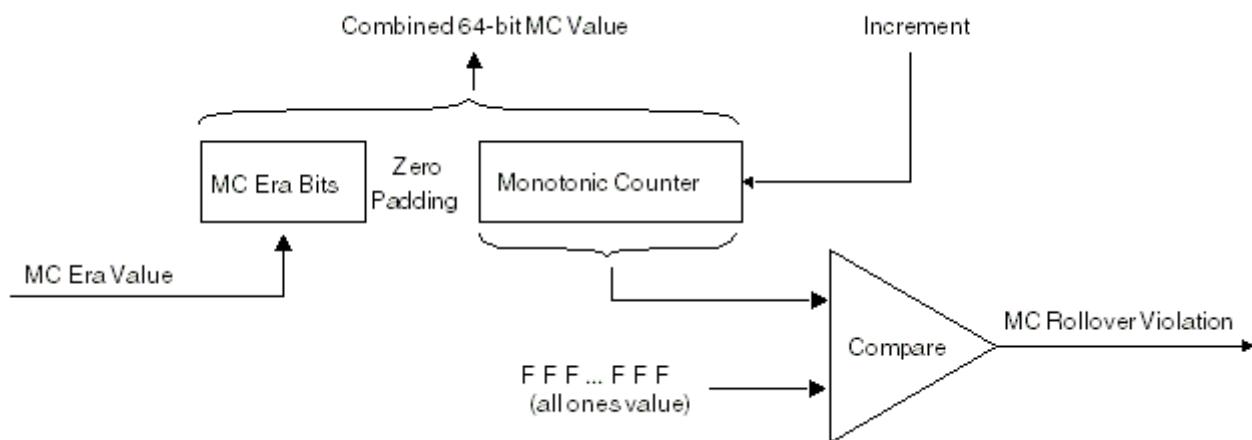


**Figure 6-7. SNVS\_LP secure real time counter**

SNVS\_LP has its own 32-bit LP Time Alarm register. As illustrated in [Figure 6-7](#), this register generates an SRTC time alarm once the secure real time clock's 32 most significant bits match with the LP Time Alarm register. The time alarm can generate an interrupt to alert the host processor and can wake the host processor from one of its low-power modes (wait, doze, stop). This alarm can also wake up the entire system in the power-down mode by asserting the wake-up external output signal.

#### 6.4.4 Monotonic counter (MC)

The following figure shows the MC and its rollover security violation.



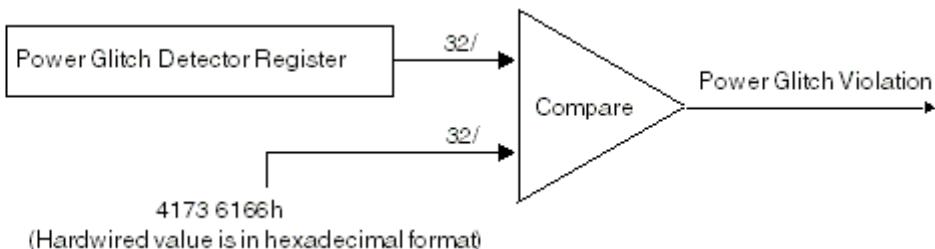
**Figure 6-8. SNVS \_LP monotonic counter**

Some security applications require a monotonic counter (MC) that cannot be exhausted or returned to any previous value during the product's lifetime. Because the MC can never repeat a number, it cannot be reset or cycled back to its starting count. If it reaches its maximum value, it does not rollover. Instead, a monotonic counter rollover indication is generated to the SNVS \_LP tamper monitor. This generates a security violation to the SSM and the host processor, which prevents a hacker from virtually running the MC backwards.

The SNVS uses an ERA value derived from the OTP elements as a mechanism for recovery from an MC failure (for example, due to a failure of LP power) where the MC value was compromised or cleared. The ERA value is prepended to the MC to form its most significant bits. Once any of the ERA value bits are set, the MC can count up from any value, including zero. This guarantees that any future value of the combined monotonic counter will be greater than any of its past values.

#### 6.4.5 Power glitch detector (PGD)

The following figure shows the PGD mechanism.

**Figure 6-9. Power glitch detector**

SNVS\_LP incorporates a PGD mechanism to detect an attack in which someone power gates the SNVS\_LP power supply for short periods to cause the LP control, status, and secure counter values to change. This mechanism detects and alerts the system whenever power supply glitches endanger SNVS\_LP register values. The mechanism works as follows:

1. The PGD register (LPPGDR) is loaded to the known specific value 4173 6166h as part of the SNVS initialization process.
2. This register's value is compared to a hardwired value to detect a glitch in the LP power supply.

Power glitch detection is always enabled and cannot be disabled. Immediately after LP POR, this register is cleared and a power glitch violation is asserted. Therefore, before programming any feature in the SNVS, users should set a proper value into LPPGDR (see [SNVS\\_LP Power Glitch Detector Register \(SNVS\\_LPPGDR\)](#)) and should clear the power glitch record in the LP status register (see [SNVS\\_LP Status Register \(SNVS\\_LPSR\)](#)).

#### 6.4.6 General-Purpose Register

The 32-bit general-purpose register allows data storage during system power-down mode as long as the SNVS\_LP remains powered.

#### 6.4.7 LP security violation/tamper policy

The following table describes SNVS\_LP security violation sources.

**Table 6-7. SNVS\_LP security violation sources**

Security violation/tamper source	Default behavior	Configuration options	Comments
Internal to LP Sources			

*Table continues on the next page...*

**Table 6-7. SNVS \_LP security violation sources  
(continued)**

Security violation/tamper source	Default behavior	Configuration options	Comments
SRTC Rollover Violation	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when SRTC is enabled and it reaches maximal value (all ones)
MC Rollover Violation	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when MC is enabled and it reaches maximal value (all ones)
Power Glitch Violation	Enable	-	Asserted when value in the Power Glitch Detector Register is different from the predefined hardwired value
SRTC Clock Tampering	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when Clock Monitor generates "No Clock" violation
Temperature Tamper	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when Temperature Monitor generates "Out of Range Temperature" violation
Voltage Tamper	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when Voltage Monitor generates "Out of Range Voltage" violation
Wire-Mesh Tamper	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted when analog detector generates wire-mesh tamper indication
External Tamper	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted on the external tamper pin
External Tamper 2	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted on the external tamper pin 2
From HP Section			
LP software violation	Enable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted by software
HP Security Violation Inputs 0-5	Disable	<ul style="list-style-type: none"> <li>• Enable</li> <li>• Disable</li> </ul>	Asserted on the Security Violation Port

Once generated, an LP security violation is reported to SNVS \_HP. The source of the violation is recorded in the LP Status Register. An LP security violation clears the ZMK registers.

## 6.5 Modes of operation

The SNVS operates in either the system power-down or system power-up mode of operation.

During system power-down, SNVS \_HP is powered-down. SNVS \_LP is powered from the backup power supply and is electrically isolated from the rest of the chip. In this mode, SNVS \_LP keeps its registers' values and monitors the SNVS \_LP tamper detection inputs.

During system power-up, SNVS \_HP and SNVS \_LP are both powered-up and all SNVS functions are operational.

### 6.5.1 Operational states

The SNVS incorporates a system security monitor (SSM) module, which is responsible for monitoring system security violations and controlling security state of the system. The SSM states are defined as follows:

- Init.—System is powered up after system POR
- Check—System performs security checks
- Non-Secure (Functional)—System operates in non-secure state
- Trusted (Functional)—System operates in trusted state
- Secure (Functional)—System operates in secure state
- Soft Fail—Security violation/tamper was detected. Command to clear sensitive data is generated
- Hard Fail—System hard reset is requested

Three of the SSM states—trusted, secure, and non-secure—are considered functional states, meaning SSM states during which CAAM and the rest of the chip's functional blocks are expected to operate normally.

## 6.6 SNVS clock sources

**Table 6-8. SNVS clock sources**

Clock	Description	Synchronization requirements
System peripheral clock	<ul style="list-style-type: none"> <li>• Used by the SNVS internal logic, such as the system security monitor.</li> <li>• Can be gated outside of the module when SNVS indicates that it is not in use.</li> </ul>	This is the main clock with respect to which the System IP bus access clock is synchronized.
System IP bus access clock	<ul style="list-style-type: none"> <li>• Used by the SNVS for clocking its registers during read/write accesses.</li> <li>• Active only during the IP Bus access cycle.</li> </ul>	Synchronized with the system peripheral clock.
LP SRTC clock	<ul style="list-style-type: none"> <li>• Used by the Secure Real Time Counter.</li> <li>• Clock input should be driven directly from an external clock source (through the chip pin) without any intervening logic.</li> </ul>	Does not have to be synchronous with the other clocks.
HP RTC clock	<ul style="list-style-type: none"> <li>• Used by SNVS _HP real time counter.</li> </ul>	Does not have to be synchronous with the other clocks.

## 6.7 SNVS reset and system power up

Table 6-9. Reset summary

Reset	Source	Characteristics	Internally resets
HP Hard	ipg_hard_async_reset_b	active-low, asynchronous	All SNVS _HP SNVS _LP registers and flops.
LP Power On Reset (POR)	lp_por_b	active-low, asynchronous	All SNVS _LP registers and flops
Scan Enter	T-Secure	active-high, synchronous, 2 cycles	The following registers/flops are cleared: <ul style="list-style-type: none"> <li>• ZMK Registers and ZMK_VAL bit</li> <li>• SRTC_ENV bit</li> <li>• MC_ENV bit</li> </ul>
Scan Exit	T-Secure	active-high, synchronous	All SNVS _LP registers except one Scan Exit status bit (SED) of the SNVS _LP Status Register (LPSR)
Field Return System Configuration	System Security Configuration Input	active-high	ZMK Registers and ZMK_VAL bit. This reset is permanently active in Field Return Configuration
LP software Reset	software	active-high, synchronous, 1 cycle	All SNVS _LP registers and flops. LP software Reset can be asserted if not disabled.
LP Security Violation	LP Tamper Monitor	active-high, asynchronous	The following registers/flops are cleared: <ul style="list-style-type: none"> <li>• ZMK Registers and ZMK_VAL bit</li> <li>• SRTC_ENV bit</li> <li>• MC_ENV bit</li> </ul>
ZMK Programming Mode Change	LP Control Register	active-high, synchronous, 1 cycle	The ZMK Registers and ZMK_VAL bit. This reset is asserted when a zero-to-one or one-to-zero transition occurs on the ZMK_HWP bit.

## 6.8 SNVS interrupts, alarms, and security violations

SNVS provides the following interrupt and alarm lines:

- Functional interrupt (active low)
- Security interrupt (active low)
- Wake-up alarm (active high)
- Security violation (active high)
- Hard failure violation (active high)

## Programming Guidelines

The security violation is triggered immediately upon an SSM transition to the soft fail state. Note that its assertion can be asynchronous as a result of an asynchronous LP tamper event. The SNVS hard failure indication is triggered immediately upon an SSM transition to the hard fail state.

The following table summarizes all SNVS interrupts and alarm sources.

**Table 6-10. Interrupts and alarms summary**

Interrupt/violation	Source	Default configuration	Configuration options
SNVS functional interrupt	SRTC time alarm	Disable	Enable/Disable
	RTC time alarm	Disable	Enable/Disable
	RTC periodic interrupt	Disable	Enable/Disable
SNVS security interrupt	Security violation inputs	Disable	Enable/Disable
	SSM transition to the soft fail state	Enable	-
	LP security violation	Disable	Enable/Disable
SNVS wake-up alarm	SRTC time alarm	Disable	Enable/Disable
	LP security violation	Disable	Enable/Disable
SNVS security violation	SSM transition to the soft fail state	Enable	-
SNVS hard failure violation	SSM transition to the hard fail state	Enable	-

## 6.9 Programming Guidelines

This section provides initialization and application information for the SNVS module.

### 6.9.1 RTC/SRTC control bits setting

All SNVS registers are programmed from the register bus. Therefore, any changes are synchronized with the IP clock. Several registers can also change synchronously with the RTC/SRTC clock after they are programmed. To avoid IP clock and RTC/SRTC clock synchronization issues, these values can only be programmed when the corresponding function is disabled. The following table presents the list of these values with the control bit setting required for programming.

**Table 6-11. RTC/SRTC synchronized values list**

Function	Value/register	Control bit setting
HP section		

*Table continues on the next page...*

**Table 6-11. RTC/SRTC synchronized values list (continued)**

Function	Value/register	Control bit setting
HP Real Time Counter	HPRTCMR and HPRTCLR Registers	RTC_EN = 0 - HPRTCMR/HPRTCLR can be programmed RTC_EN = 1 - HPRTCMR/HPRTCLR cannot be programmed
HP Time Alarm	HPTAMR and HPTALR Registers	HPTA_EN = 0 - HPTAMR/HPTALR can be programmed HPTA_EN = 1 - HPTAMR/HPTALR cannot be programmed
HP Time Calibration Value	HPCALB_VAL Value	HPCALB_EN = 0 - HPCALB_VAL can be programmed HPCALB_EN = 1 - HPCALB_VAL cannot be programmed
LP section		
LP Secure Real Time Counter	LPRTCMR and LPRTCLR Registers	SRTC_ENV = 0 - LPRTCMR/LPRTCLR can be programmed SRTC_ENV = 1 - LPRTCMR/LPRTCLR cannot be programmed
LP Time Alarm	LPTAR Register	LPTA_EN = 0 - LPTAR can be programmed LPTA_EN = 1 - LPTAR cannot be programmed
LP Time Calibration Value	LPCALB_VAL Value	LPCALB_EN = 0 - LPCALB_VAL can be programmed LPCALB_EN = 1 - LPCALB_VAL cannot be programmed

Use the following step to program synchronized values:

1. Check the enable bit value. If set, clear it.
2. Verify that the enable bit is cleared.

There are two reasons to verify the enable bit's setting:

- Enable bit clearing does not happen immediately; it takes three IPclock cycles and two RTC/SRTC clock cycles to change the enable bit's value.
  - If the enable bit is locked for programming, it cannot be cleared.
3. Program the desired value.
  4. Set the enable bit; it takes three IP clock cycles and two RTC/SRTC clock cycles for the bit to set.

### NOTE

Incrementing the value programmed into RTC/SRTC registers by two compensates for the two RTC/SRTC clock cycle delay that is required to enable the counter.

## 6.9.2 RTC/SRTC value read

There are two scenarios when software can read corrupted values from the RTC (HPRTCMR and HPRTCLR) and SRTC (LPSRTCMR and LPSRTCLR) registers:

- The RTC and SRTC counters are incremented by the slow 32 kHz clock, which is asynchronous to the system clock. The counter value is synchronized to the system clock before software reads that. The synchronization register may capture the counter value in the middle of the counter update. In this case, it is not guaranteed that all bits are properly sampled by the synchronization register; the value read by software can be wrong.
- The RTC and SRTC value is longer than the single bus read transaction of 32-bits. Therefore, software reads two registers, each holding a portion of the counter value. After reading one of these registers but before reading the second register, both registers may update their values. In this case, the value combined by software will be incorrect.

To avoid these issues, it is strongly recommended that software perform two consecutive reads of the RTC/SRTC value:

- If two consecutive reads are similar, the value is correct.
- If two consecutive reads are different, perform two more reads.

The worst case scenario may require three sessions of two consecutive reads.

### **6.9.3 ZMK programming guidelines**

Either software or hardware can program the ZMK.

To program ZMK by software, do the following:

1. Verify that ZMK\_HWP bit is not set.
2. Verify that ZMK is not locked for write.
3. Write key value to the ZMK registers.
4. Verify that the correct key value is written.
5. Set ZMK\_VAL bit if the ZMK (or the ZMK XORed with the OTPMK) will be used by CAAM as the master key. There is no need to set this bit if the ZMK registers are only read by software.
6. (optional) Set ZMK\_ECC\_EN bit to enable ZMK error correction code verification. Software can verify that the correct nine bit codeword is generated by reading ZMK\_ECC\_VALUE field.
7. (optional) Block software read accesses to the ZMK registers and ZMK\_ECC\_VALUE field by setting ZMK Read lock bit.
8. (optional) Block software write accesses to the ZMK registers by setting ZMK Write Lock bit.
9. Set MASTER\_KEY\_SEL and MKS\_EN bits to select combination of OTPMK and ZMK to be provided to the hardware cryptographic module.

10. (optional) Block software write accesses to the MASTER\_KEY\_SEL field by setting MKS lock bit.

To program ZMK by hardware, do the following:

1. Set ZMK\_HWP bit.
2. Verify that ZMK is not locked for write.
3. Write one to the PROG\_ZMK bit.
4. Check when ZMK\_VAL bit is set. Hardware sets this bit at the end of the ZMK programming cycle.
5. (optional) Set ZMK\_ECC\_EN bit to enable ZMK error correction code verification.

Note that software cannot read the ZMK registers and ZMK\_ECC\_VALUE field in the hardware programming mode except in Fab configuration.

6. (optional) Block hardware programming option of the ZMK registers by setting ZMK Write Lock bit.
7. Set MASTER\_KEY\_SEL and MKS\_EN bits to select the combination of OTPMK and ZMK to be provided to the hardware cryptographic module.
8. (optional) Block software write accesses to the MASTER\_KEY\_SEL field by setting MKS lock bit.

#### **6.9.4 General initialization guidelines**

Complete the following steps in order to properly initialize the module:

1. Transition SSM from check state to functional state (trusted/secure/non-secure).
2. Set the correct value in the Power Glitch Detector Register.
3. Clear the power glitch record in the LP Status Register.
4. User Specific: Enable security violations and interrupts in SNVS control and configuration registers.
5. User Specific: Program SNVS general functions/configurations.
6. User Specific: Set lock bits.

#### **NOTE**

Even if ZMK is not used, set the ZMK Write Lock bit.  
Otherwise, software can program a value to the ZMK register and select this value as a master key output.

## **6.10 SNVS Memory Map/Register Definition**

## SNVS Memory Map/Register Definition

This section contains detailed register descriptions for the SNVS registers. Each description includes a standard register diagram and register table. The register table provides detailed descriptions of the register bit and field function, in bit order.

SNVS registers consist of two types:

- Privileged read/write accessible
- Non-privileged read/write accessible

Privileged read/write accessible registers can only be accessed for read/write by privileged software. Unauthorized write accesses are ignored, and unauthorized read accesses return zero. Non-privileged software can access privileged access registers when the non-privileged software access enable bit is set in the SNVS \_HP Command Register.

In addition, all privileged access registers (except HPSVSR and LPSR) can be written to only if the system security monitor is in one of the three functional states:

- Non-Secure
- Trusted
- Secure

Certain fields in the SNVS \_HP Command Register can be written in non-functional system security monitor states (init, check, soft fail, and hard fail) as follows:

- SW\_LPSV, SSM\_ST, SW\_SV, and SW\_FSV can be written in check or soft fail state.
- The HAC\_STOP bit can only be set in soft fail state but can be cleared in either soft fail or a functional state.
- HAC\_LOAD, HAC\_CLEAR bits and HPSVSR, LPSR Registers can be accessed in soft fail or a functional state.

The system security monitor state does not restrict read access to SNVS registers.

Non-privileged read/write accessible registers are read/write accessible by any software.

The following table shows the SNVS memory map. The LP register values are set only on LP POR and are unaffected by System (HP) POR. The HP registers are set only on System POR and are unaffected by LP POR.

**SNVS memory map**

Offset address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
0	SNVS_HP Lock Register (SNVS_HPLR)	32	R/W	0000_0000h	<a href="#">6.10.1/348</a>
4	SNVS_HP Command Register (SNVS_HPCOMR)	32	R/W	0000_0000h	<a href="#">6.10.2/351</a>
8	SNVS_HP Control Register (SNVS_HPCR)	32	R/W	0000_0000h	<a href="#">6.10.3/355</a>

*Table continues on the next page...*

**SNVS memory map (continued)**

Offset address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
C	SNVS_HP Security Interrupt Control Register (SNVS_HPSICR)	32	R/W	0000_0000h	<a href="#">6.10.4/357</a>
10	SNVS_HP Security Violation Control Register (SNVS_HPSVCR)	32	R/W	0000_0000h	<a href="#">6.10.5/359</a>
14	SNVS_HP Status Register (SNVS_HPSR)	32	R/W	<a href="#">See section</a>	<a href="#">6.10.6/361</a>
18	SNVS_HP Security Violation Status Register (SNVS_HPSVSR)	32	R/W	0000_0000h	<a href="#">6.10.7/363</a>
1C	SNVS_HP High Assurance Counter IV Register (SNVS_PHACIVR)	32	R/W	0000_0000h	<a href="#">6.10.8/364</a>
20	SNVS_HP High Assurance Counter Register (SNVS_PHACR)	32	R/W	0000_0000h	<a href="#">6.10.9/365</a>
24	SNVS_HP Real Time Counter MSB Register (SNVS_HPRTCMR)	32	R/W	0000_0000h	<a href="#">6.10.10/365</a>
28	SNVS_HP Real Time Counter LSB Register (SNVS_HPRTCLR)	32	R/W	0000_0000h	<a href="#">6.10.11/366</a>
2C	SNVS_HP Time Alarm MSB Register (SNVS_HPTAMR)	32	R/W	0000_0000h	<a href="#">6.10.12/366</a>
30	SNVS_HP Time Alarm LSB Register (SNVS_HPTALR)	32	R/W	0000_0000h	<a href="#">6.10.13/367</a>
34	SNVS_LP Lock Register (SNVS_LPLR)	32	R/W	0000_0000h	<a href="#">6.10.14/368</a>
38	SNVS_LP Control Register (SNVS_LPCR)	32	R/W	0000_0000h	<a href="#">6.10.15/370</a>
3C	SNVS_LP Master Key Control Register (SNVS_LPMKCR)	32	R/W	0000_0000h	<a href="#">6.10.16/372</a>
40	SNVS_LP Security Violation Control Register (SNVS_LPSVCR)	32	R/W	0000_0000h	<a href="#">6.10.17/374</a>
44	SNVS_LP Tamper Glitch Filters Configuration Register (SNVS_LPTGFCR)	32	R/W	0000_0000h	<a href="#">6.10.18/376</a>
48	SNVS_LP Tamper Detectors Configuration Register (SNVS_LPTDCR)	32	R/W	0000_0000h	<a href="#">6.10.19/378</a>
4C	SNVS_LP Status Register (SNVS_LPSR)	32	R/W	0000_0008h	<a href="#">6.10.20/381</a>
50	SNVS_LP Secure Real Time Counter MSB Register (SNVS_LPSRTCMR)	32	R/W	0000_0000h	<a href="#">6.10.21/384</a>
54	SNVS_LP Secure Real Time Counter LSB Register (SNVS_LPSRTCLR)	32	R/W	0000_0000h	<a href="#">6.10.22/385</a>
58	SNVS_LP Time Alarm Register (SNVS_LPTAR)	32	R/W	0000_0000h	<a href="#">6.10.23/385</a>
5C	SNVS_LP Secure Monotonic Counter MSB Register (SNVS_LPSMCMR)	32	R/W	0000_0000h	<a href="#">6.10.24/386</a>
60	SNVS_LP Secure Monotonic Counter LSB Register (SNVS_LPSMCLR)	32	R/W	0000_0000h	<a href="#">6.10.25/386</a>
64	SNVS_LP Power Glitch Detector Register (SNVS_LPPGDR)	32	R/W	0000_0000h	<a href="#">6.10.26/387</a>
68	SNVS_LP General Purpose Register (SNVS_LPGPR)	32	R/W	0000_0000h	<a href="#">6.10.27/387</a>
6C	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR0)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
70	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR1)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>

Table continues on the next page...

**SNVS memory map (continued)**

Offset address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
74	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR2)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
78	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR3)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
7C	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR4)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
80	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR5)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
84	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR6)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
88	SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKR7)	32	R/W	0000_0000h	<a href="#">6.10.28/388</a>
BF8	SNVS_HP Version ID Register 1 (SNVS_HPV IDR1)	32	R	<a href="#">See section</a>	<a href="#">6.10.29/389</a>
BFC	SNVS_HP Version ID Register 2 (SNVS_HPV IDR2)	32	R	0000_0000h	<a href="#">6.10.30/389</a>

**6.10.1 SNVS\_HP Lock Register (SNVS\_HPLR)**

The SNVS \_HP Lock Register contains lock bits for the SNVS registers. This is a privileged write register.

Address: 0h base + 0h offset = 0h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SNVS\_HPLR field descriptions**

Field	Description
31–19 -	This field is reserved. Reserved
18 HAC_L	High Assurance Configuration Lock  When set, prevents any writes to HPHACIVR, HPHACR, and HAC_EN bit of HPCOMR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
17 HPSICR_L	HP Security Interrupt Control Register Lock  When set, prevents any writes to the HPSICR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
16 HPSVCR_L	HP Security Violation Control Register Lock  When set, prevents any writes to the HPSVCR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
15–10 -	This field is reserved. Reserved
9 MKS_SL	Master Key Select Soft Lock  When set, prevents any writes to the MASTER_KEY_SEL field of the LPMKCR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
8 LPTDCR_SL	LP Tamper Detectors Configuration Register Soft Lock  When set, prevents any writes to the LPTDCR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
7 LPTGFCR_SL	LP Tamper Glitch Filter Configuration Register Soft Lock  When set, prevents any writes to the LPTGFCR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
6 LPSVCR_SL	LP Security Violation Control Register Soft Lock  When set, prevents any writes to the LPSVCR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed
5 GPR_SL	General Purpose Register Soft Lock  When set, prevents any writes to the GPR. Once set, this bit can only be reset by the system reset.  0 Write access is allowed 1 Write access is not allowed

*Table continues on the next page...*

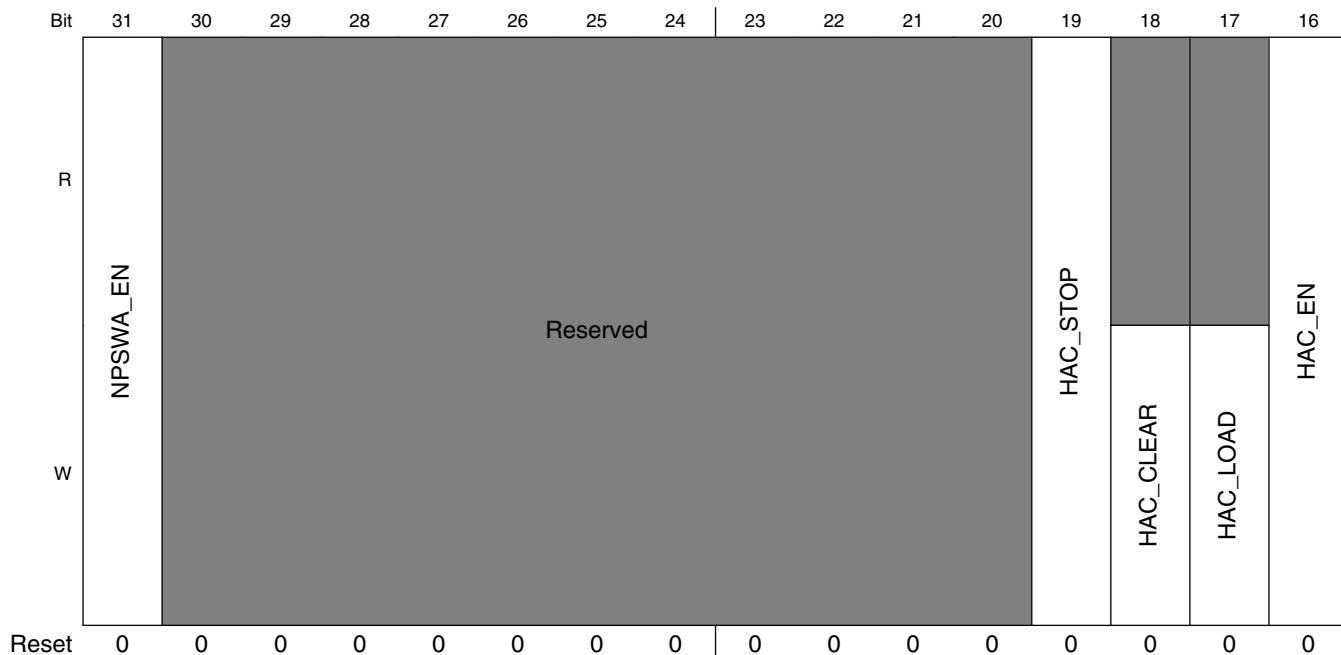
**SNVS\_HPLR field descriptions (continued)**

Field	Description
4 MC_SL	<p>Monotonic Counter Soft Lock</p> <p>When set, prevents any writes (increments) to the MC Registers and MC_ENV bit. Once set, this bit can only be reset by the system reset.</p> <p>0 Write access (increment) is allowed 1 Write access (increment) is not allowed</p>
3 LPCALB_SL	<p>LP Calibration Soft Lock</p> <p>When set, prevents any writes to the LP Calibration Value (LPCALB_VAL) and LP Calibration Enable (LPCALB_EN). Once set, this bit can only be reset by the system reset.</p> <p>0 Write access is allowed 1 Write access is not allowed</p>
2 SRTC_SL	<p>Secure Real Time Counter Soft Lock</p> <p>When set, prevents any writes to the SRTC Registers, SRTC_ENV, and SRTC_INV_EN bits. Once set, this bit can only be reset by the system reset.</p> <p>0 Write access is allowed 1 Write access is not allowed</p>
1 ZMK_RSL	<p>Zeroizable Master Key Read Soft Lock</p> <p>When set, prevents any software reads to the ZMK Registers and ZMK_ECC_VALUE field of the LPMKCR. In ZMK hardware programming mode (ZMK_HWP is set), the ZMK and ZMK_ECC_VALUE cannot be read by software. Regardless of the bit setting, hardware can use the ZMK value when ZMK is selected. Once set, this bit can only be reset by the system reset.</p> <p>0 Read access is allowed (only in software Programming mode) 1 Read access is not allowed</p>
0 ZMK_WSL	<p>Zeroizable Master Key Write Soft Lock</p> <p>When set, prevents any writes (software and hardware) to the ZMK registers and MASTER_KEY_SEL, ZMK_HWP, ZMK_VAL, and ZMK_ECC_EN fields of the LPMKCR. Once set, this bit can only be cleared by system reset.</p> <p>0 Write access is allowed 1 Write access is not allowed</p>

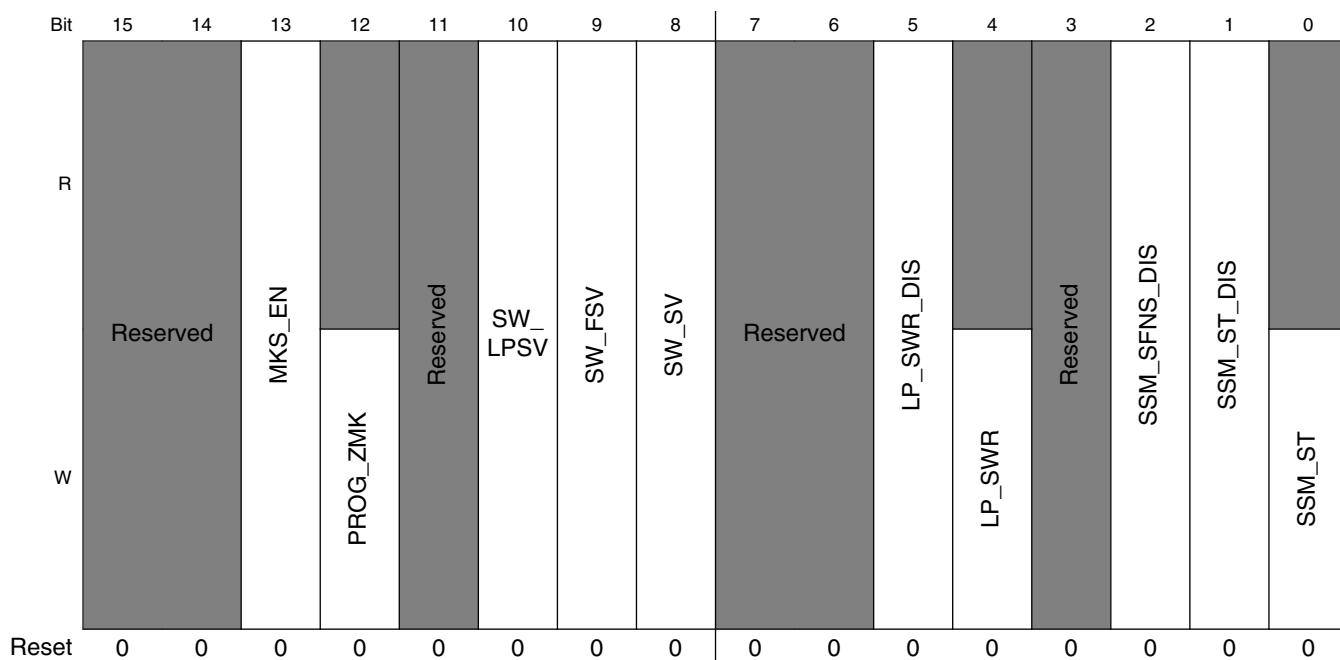
## 6.10.2 SNVS\_HP Command Register (SNVS\_HPCOMR)

The SNVS \_HP Command Register contains the command, configuration, and control bits for the SNVS block. Some fields of this register can be written to in check and soft fail states in addition to the standard write access in functional states. This is a privileged write register.

Address: 0h base + 4h offset = 4h



## SNVS Memory Map/Register Definition



### SNVS\_HPCOMR field descriptions

Field	Description
31 NPSWA_EN	<p>Non-Privileged Software Access Enable</p> <p>When set, allows non-privileged software to access all SNVS registers, including those that are privileged software read/write access only.</p> <p>0 Only privileged software can access privileged registers</p> <p>1 Any software can access privileged registers</p>
30–20 -	This field is reserved. Reserved
19 HAC_STOP	<p>High Assurance Counter Stop</p> <p>This bit can be set only when SSM is in soft fail state. When set, it stops the high assurance counter and prevents transition to the hard fail state. This bit can be cleared in a functional or soft fail state. If the bit is cleared in the soft fail state, the high assurance counter counts down from the place where it was stopped.</p> <p>0 HAC counter can count down</p> <p>1 HAC counter is stopped</p>
18 HAC_CLEAR	<p>High Assurance Counter Clear</p> <p>When set, it clears the High Assurance Counter Register. It can be cleared in a functional or soft fail state. If the HAC counter is cleared in the soft fail state, the SSM transitions to the hard fail state if high assurance configuration is enabled (HAC_EN is set). This self-clearing bit is always read as zero.</p> <p>0 No Action</p> <p>1 Clear the HAC</p>
17 HAC_LOAD	<p>High Assurance Counter Load</p> <p>When set, it loads the High Assurance Counter Register with the value of the High Assurance Counter Load Register. It can be done in a functional or soft fail state. This self-clearing bit is always read as zero.</p>

Table continues on the next page...

**SNVS\_HPCOMR field descriptions (continued)**

Field	Description
	<p>0 No Action 1 Load the HAC</p>
16 HAC_EN	<p><b>High Assurance Configuration Enable</b> This bit controls the SSM transition from the soft fail to the hard fail state. When this bit is set and software fails to stop the HAC before it expires, the SSM transitions to the hard fail state. This bit cannot be changed once HAC_L bit is set.</p> <p>0 High Assurance Configuration is disabled 1 High Assurance Configuration is enabled</p>
15–14 -	This field is reserved. Reserved
13 MKS_EN	<p><b>Master Key Select Enable</b> When not set, the one time programmable (OTP) master key is selected by default. When set, the master key is selected according to the setting of the master key select field (MASTER_KEY_SEL) of LPMKCR. Once set, this bit can only be reset by the system reset.</p> <p>0 OTP master key is selected as an SNVS master key 1 SNVS master key is selected according to the setting of the MASTER_KEY_SEL field of LPMKCR</p>
12 PROG_ZMK	<p><b>Program Zeroizable Master Key</b> This bit activates ZMK hardware programming mechanism. This mechanism is activated only if the ZMK is configured to the hardware programming mode and ZMK is not locked for writes. This self-clearing bit is always read as zero.</p> <p>0 No Action 1 Activate hardware key programming mechanism</p>
11 -	This field is reserved. Reserved
10 SW_LPSV	<p><b>LP Software Security Violation</b> When set, SNVS_LP treats this bit as a security violation. The LP secure data is zeroized or invalidated according to the configuration. This security violation may result in a system security monitor transition if the LP Security Violation is enabled in the SNVS_HP Security Violation Control Register.</p>
9 SW_FSV	<p><b>Software Fatal Security Violation</b> When set, the system security monitor treats this bit as a fatal security violation. This security violation has no effect on the LP section. This command results only in the following transitions of the SSM:</p> <p>Check State → Soft Fail Non-Secure State → Soft Fail Trusted State → Soft Fail Secure State → Soft Fail</p>
8 SW_SV	<p><b>Software Security Violation</b> When set, the system security monitor treats this bit as a non-fatal security violation. This security violation has no effect on the LP section. This command results only in the following transitions of the SSM:</p> <p>Check → Non-Secure Trusted → Soft Fail Secure → Soft Fail</p>

*Table continues on the next page...*

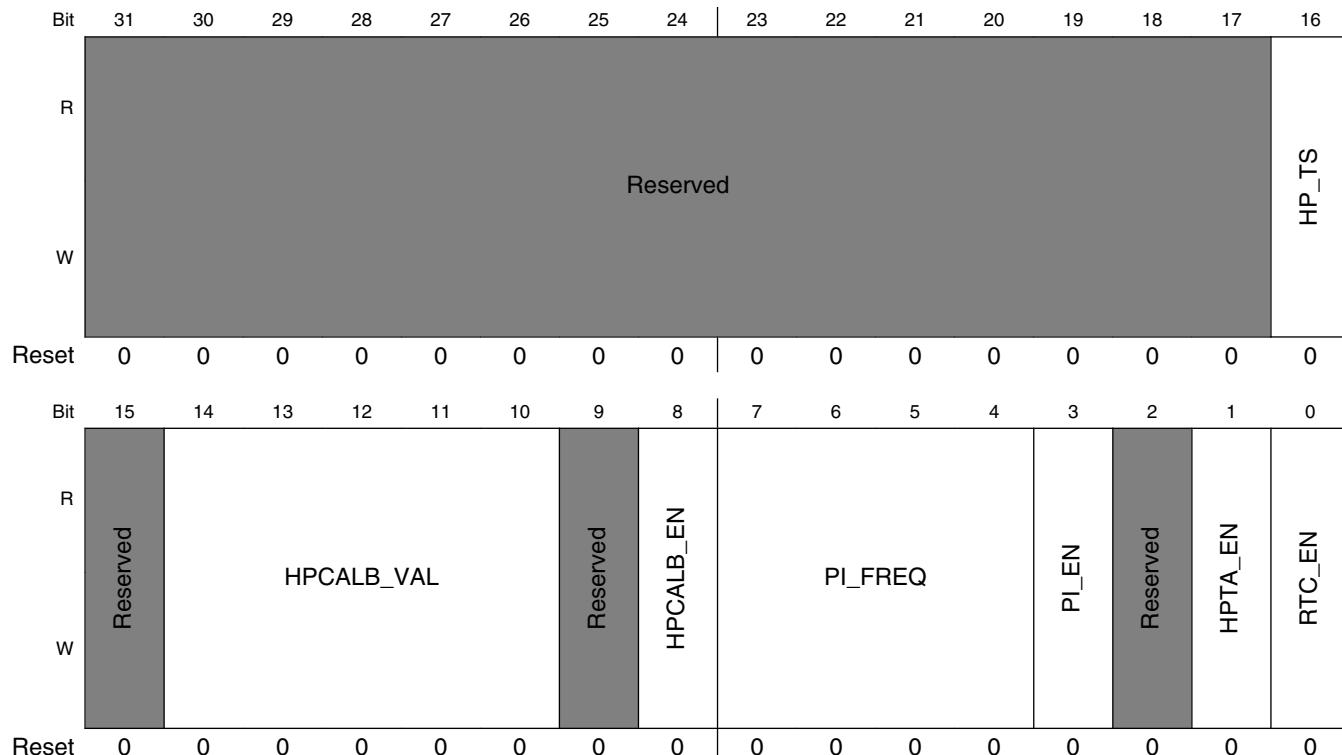
**SNVS\_HPCOMR field descriptions (continued)**

Field	Description
7–6 -	This field is reserved. Reserved
5 LP_SWR_DIS	LP Software Reset Disable  When set, disables the LP software reset. Once set, this bit can only be reset by the system reset.  0 LP software reset is enabled 1 LP software reset is disabled
4 LP_SWR	LP Software Reset  When set, it resets the SNVS_LP section. This bit cannot be set when the LP_SWR_DIS bit is set. This self-clearing bit is always read as zero.  0 No Action 1 Reset LP section
3 -	This field is reserved. Reserved
2 SSM_SFNS_DIS	SSM Soft Fail to Non-Secure State Transition Disable  When set, it disables the SSM transition from soft fail to non-secure state. Once set after the reset this bit cannot be changed  0 Soft Fail to Non-Secure State transition is enabled 1 Soft Fail to Non-Secure State transition is disabled
1 SSM_ST_DIS	SSM Secure to Trusted State Transition Disable  When set, disables the SSM transition from secure to trusted state. Once set after the reset, this bit cannot be changed.  0 Secure to Trusted State transition is enabled 1 Secure to Trusted State transition is disabled
0 SSM_ST	SSM State Transition  Transition state of the system security monitor. This self-clearing bit is always read as zero. This command results only in the following transitions of the SSM:  Check State → Non-Secure (when Non-Secure Boot and not in Fab Configuration ) Check State → Trusted (when Secure Boot or in Fab Configuration ) Trusted State → Secure Secure State → Trusted (if not disabled by SSM_ST_DIS bit) Soft Fail → Non-Secure (if not disabled by SSM_SFNS_DIS bit)

### 6.10.3 SNVS\_HP Control Register (SNVS\_HPCR)

The SNVS \_HP Control Register contains various control bits of the HP section of SNVS .

Address: 0h base + 8h offset = 8h



**SNVS\_HPCR field descriptions**

Field	Description
31–17 -	This field is reserved. Reserved
16 HP_TS	HP Time Synchronize When set, this updates the HP Time Counter with the LP Time Counter value. This self-clearing bit is always read as zero. 0 No Action 1 Synchronize the HP Time Counter to the LP Time Counter
15 -	This field is reserved. Reserved
14–10 HPCALB_VAL	HP Calibration Value

*Table continues on the next page...*

**SNVS\_HPCR field descriptions (continued)**

Field	Description
	Defines signed calibration value for the HP Real Time Counter. This field can be programmed only when RTC Calibration is disabled (HPCALB_EN is not set). This is a 5-bit 2's complement value. Hence, the allowable calibration values are in the range from -16 to +15 counts per 32768 ticks of the counter.  00000 +0 counts per each 32768 ticks of the counter 00001 +1 counts per each 32768 ticks of the counter 00010 +2 counts per each 32768 ticks of the counter 01111 +15 counts per each 32768 ticks of the counter 10000 -16 counts per each 32768 ticks of the counter 10001 -15 counts per each 32768 ticks of the counter 11110 -2 counts per each 32768 ticks of the counter 11111 -1 counts per each 32768 ticks of the counter
9 -	This field is reserved. Reserved
8 HPCALB_EN	HP Real Time Counter Calibration Enabled  Indicates that the time calibration mechanism is enabled.  0 HP Timer calibration disabled 1 HP Timer calibration enabled
7-4 PI_FREQ	Periodic Interrupt Frequency  Defines frequency of the periodic interrupt. The interrupt is generated when a zero-to-one or one-to-zero transition occurs on the selected bit of the HP Real Time Counter and Real Time Counter and Periodic Interrupt are both enabled (RTC_EN and PI_EN are set). It is recommended to program this field when Periodic Interrupt is disabled (PI_EN is not set). The possible frequencies are:  0000 - bit 0 of the RTC is selected as a source of the periodic interrupt 0001 - bit 1 of the RTC is selected as a source of the periodic interrupt 0010 - bit 2 of the RTC is selected as a source of the periodic interrupt 0011 - bit 3 of the RTC is selected as a source of the periodic interrupt 0100 - bit 4 of the RTC is selected as a source of the periodic interrupt 0101 - bit 5 of the RTC is selected as a source of the periodic interrupt 0110 - bit 6 of the RTC is selected as a source of the periodic interrupt 0111 - bit 7 of the RTC is selected as a source of the periodic interrupt 1000 - bit 8 of the RTC is selected as a source of the periodic interrupt 1001 - bit 9 of the RTC is selected as a source of the periodic interrupt 1010 - bit 10 of the RTC is selected as a source of the periodic interrupt 1011 - bit 11 of the RTC is selected as a source of the periodic interrupt 1100 - bit 12 of the RTC is selected as a source of the periodic interrupt 1101 - bit 13 of the RTC is selected as a source of the periodic interrupt 1110 - bit 14 of the RTC is selected as a source of the periodic interrupt 1111 - bit 15 of the RTC is selected as a source of the periodic interrupt
3 PI_EN	HP Periodic Interrupt Enable  The periodic interrupt can be generated only if the HP Real Time Counter is enabled.  0 HP Periodic Interrupt is disabled 1 HP Periodic Interrupt is enabled
2 -	This field is reserved. Reserved

*Table continues on the next page...*

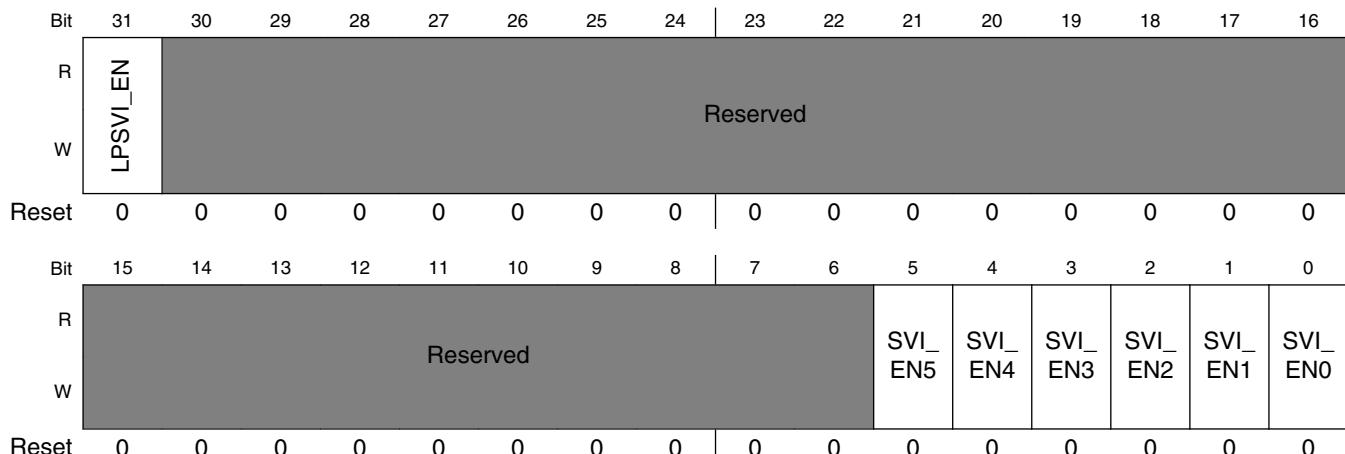
**SNVS\_HPCR field descriptions (continued)**

Field	Description
1 HPTA_EN	<p>HP Time Alarm Enable</p> <p>When set, the time alarm interrupt is generated if the value in the HP Time Alarm Registers is equal to the value of the HP Real Time Counter.</p> <p>0 HP Time Alarm Interrupt is disabled 1 HP Time Alarm Interrupt is enabled</p>
0 RTC_EN	<p>HP Real Time Counter Enable</p> <p>0 RTC is disabled 1 RTC is enabled</p>

**6.10.4 SNVS\_HP Security Interrupt Control Register (SNVS\_HPSICR)**

The HP Security Interrupt Control Register defines the SNVS security interrupt generation policy. This is a privileged write register.

Address: 0h base + Ch offset = Ch

**SNVS\_HPSICR field descriptions**

Field	Description
31 LPSVI_EN	<p>LP Security Violation Interrupt Enable</p> <p>This bit enables generating of the security interrupt to the host processor upon security violation signal from the LP section.</p> <p>0 LP Security Violation Interrupt is Disabled 1 LP Security Violation Interrupt is Enabled</p>
30–6 -	This field is reserved. Reserved

*Table continues on the next page...*

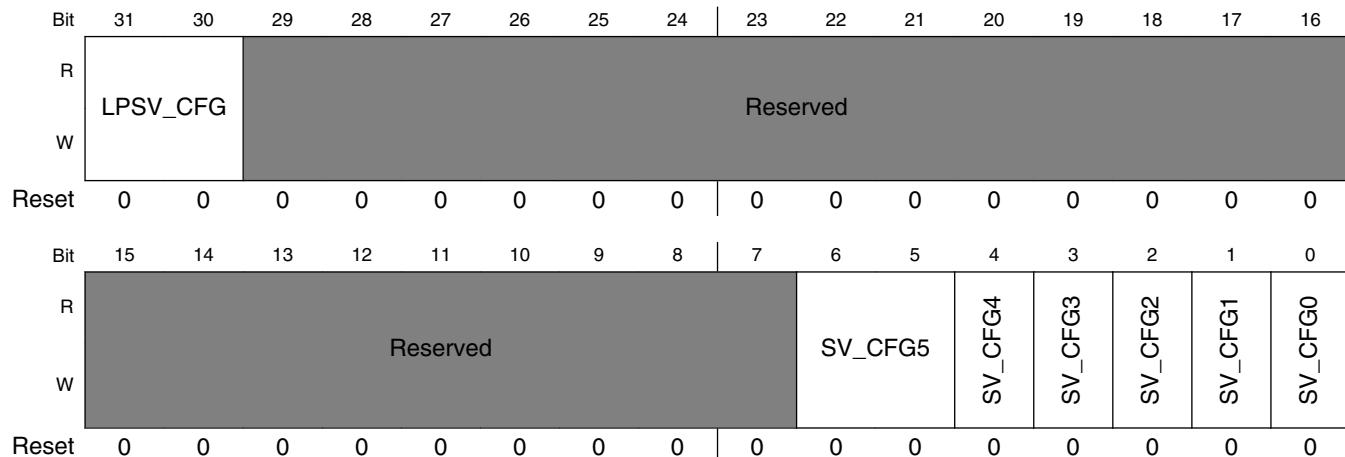
**SNVS\_HPSICR field descriptions (continued)**

Field	Description
5 SVI_EN5	<p>Security Violation Interrupt 5 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 5.</p> <p>0 Security Violation Interrupt 5 is Disabled 1 Security Violation Interrupt 5 is Enabled</p>
4 SVI_EN4	<p>Security Violation Interrupt 4 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 4.</p> <p>0 Security Violation Interrupt 4 is Disabled 1 Security Violation Interrupt 4 is Enabled</p>
3 SVI_EN3	<p>Security Violation Interrupt 3 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 3.</p> <p>0 Security Violation Interrupt 3 is Disabled 1 Security Violation Interrupt 3 is Enabled</p>
2 SVI_EN2	<p>Security Violation Interrupt 2 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 2.</p> <p>0 Security Violation Interrupt 2 is Disabled 1 Security Violation Interrupt 2 is Enabled</p>
1 SVI_EN1	<p>Security Violation Interrupt 1 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 1.</p> <p>0 Security Violation Interrupt 1 is Disabled 1 Security Violation Interrupt 1 is Enabled</p>
0 SVI_EN0	<p>Security Violation Interrupt 0 Enable</p> <p>This bit enables generation of the security interrupt to the host processor upon security violation detection on input port 0.</p> <p>0 Security Violation Interrupt 0 is Disabled 1 Security Violation Interrupt 0 is Enabled</p>

## 6.10.5 SNVS\_HP Security Violation Control Register (SNVS\_HPSVCR)

The HP Security Violation Control Register defines types for each security violation input. This is a privileged write register.

Address: 0h base + 10h offset = 10h



### SNVS\_HPSVCR field descriptions

Field	Description
31–30 LPSV_CFG	LP Security Violation Configuration This field configures the LP security violation source. 00 LP security violation is disabled 01 LP security violation is a non-fatal violation 1x LP security violation is a fatal violation
29–7 -	This field is reserved. Reserved
6–5 SV_CFG5	Security Violation Input 5 Configuration This field configures the Security Violation Input 5. This setting instructs the SSM how to respond when a security violation on port 5 is detected. 00 Security Violation 5 is disabled 01 Security Violation 5 is a non-fatal violation 1x Security Violation 5 is a fatal violation
4 SV_CFG4	Security Violation Input 4 Configuration This field configures the security violation Input 4. This setting instructs the SSM how to respond when a security violation on port 4 is detected. 0 Security Violation 4 is a non-fatal violation 1 Security Violation 4 is a fatal violation

Table continues on the next page...

**SNVS\_HPSVCR field descriptions (continued)**

Field	Description
3 SV_CFG3	<p>Security Violation Input 3 Configuration</p> <p>This field configures the security violation input 3. This setting instructs the SSM how to respond when a security violation on port 3 is detected.</p> <ul style="list-style-type: none"> <li>0 Security Violation 3 is a non-fatal violation</li> <li>1 Security Violation 3 is a fatal violation</li> </ul>
2 SV_CFG2	<p>Security Violation Input 2 Configuration</p> <p>This field configures the security violation input 2. This setting instructs the SSM how to respond when a security violation on port 2 is detected.</p> <ul style="list-style-type: none"> <li>0 Security Violation 2 is a non-fatal violation</li> <li>1 Security Violation 2 is a fatal violation</li> </ul>
1 SV_CFG1	<p>Security Violation Input 1 Configuration</p> <p>This field configures the Security Violation Input 1. This setting instructs the SSM how to respond when a security violation on port 1 is detected.</p> <ul style="list-style-type: none"> <li>0 Security Violation 1 is a non-fatal violation</li> <li>1 Security Violation 1 is a fatal violation</li> </ul>
0 SV_CFG0	<p>Security Violation Input 0 Configuration</p> <p>This field configures the security violation input 0. This setting instructs the SSM how to respond when a security violation on port 0 is detected.</p> <ul style="list-style-type: none"> <li>0 Security Violation 0 is a non-fatal violation</li> <li>1 Security Violation 0 is a fatal violation</li> </ul>

## 6.10.6 SNVS\_HP Status Register (SNVS\_HPSR)

The HP Status Register reflects the internal state of the SNVS .

Address: 0h base + 14h offset = 14h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ZMK_ZERO	Reserved				OTPMK_ZERO	Reserved				OTPMK_SYNDROME					
W																
Reset	1*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYS_SECURE_BOOT	SYS_SECURITY_CFG				SSM_ST				Reserved			LPDIS	Reserved	PI	HPTA
W																
Reset	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*

\* Notes:

- The reset value depends on the value of the LPZMKR registers and the value of the OTPMK programmed in fuses

### SNVS\_HPSR field descriptions

Field	Description
31 ZMK_ZERO	Zeroizable Master Key is Equal to Zero. When set, this bit triggers "bad key" violation if the ZMK is selected for use  0 The ZMK is not zero. 1 The ZMK is zero.
30–28 -	This field is reserved. Reserved
27 OTPMK_ZERO	One Time Programmable Master Key is Equal to Zero. When set, this bit always triggers "bad key" violation  0 The OTPMK is not zero. 1 The OTPMK is zero.
26–25 -	This field is reserved. Reserved
24–16 OTPMK_SYNDROME	One Time Programmable Master Key Syndrome Value  The eight lower bits of this value indicate error location in case of a single-bit error. For example, syndrome word 10010110 indicates that key bit 150 has an error.

Table continues on the next page...

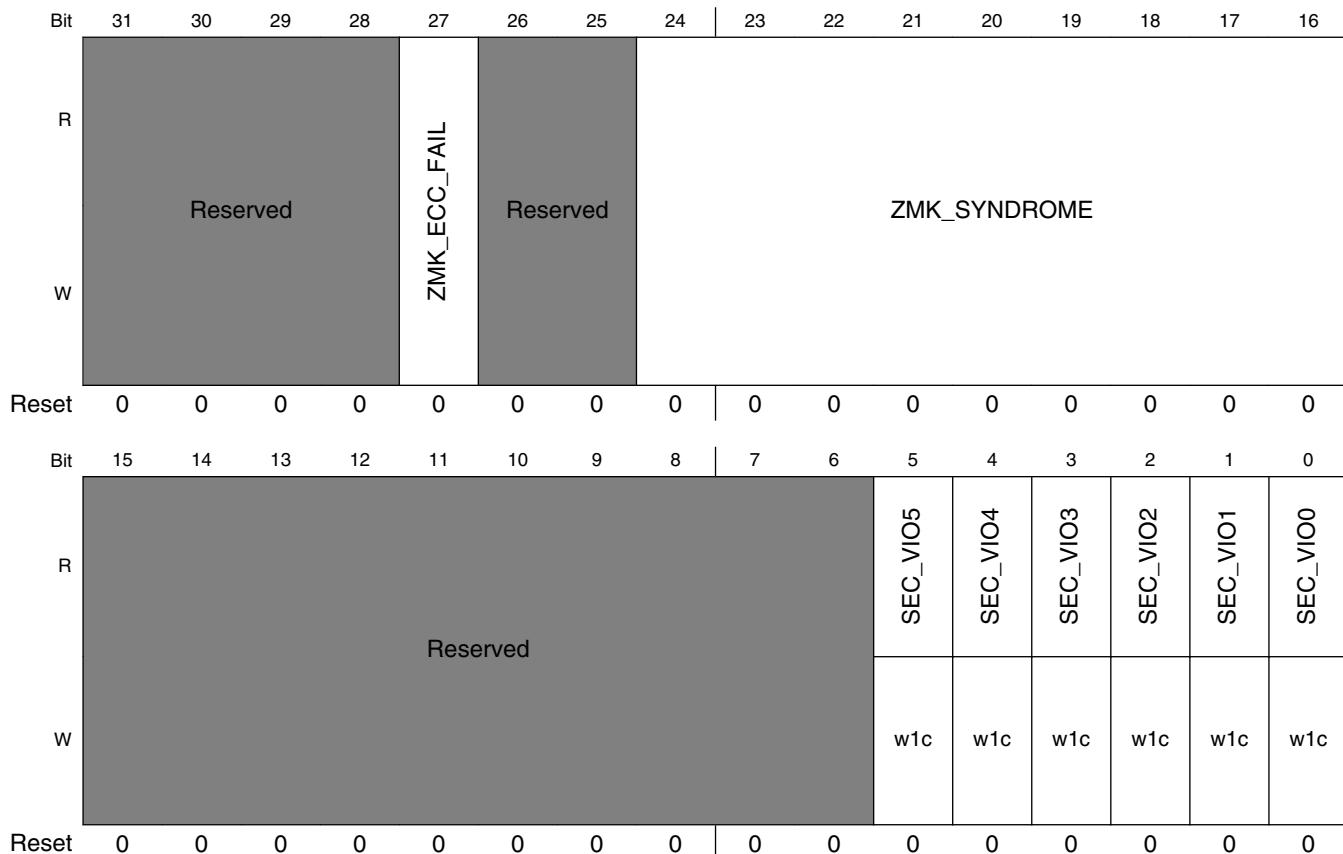
**SNVS\_HPSR field descriptions (continued)**

Field	Description
	The ninth bit of the syndrome word checks parity of the whole key value. This bit is 1 when the odd number of errors are occurred and it is 0 when the number of errors is even. For example, if one of the eight bits indicates a failure and the ninth bit is zero then the number of errors in the one time programmable master key is at least 2 and it cannot be corrected. When one of the syndrome bits is set, the bad key violation is always generated
15 SYS_SECURE_BOOT	This bit reflects the value of the sys_secure_boot input signal to SNVS . If this bit is 1, the chip boots from internal ROM.
14–12 SYS_SECURITY_CFG	This field reflects the value of the sys_security_cfg input signal, which is defined as follows: 000 Fab Configuration - the default configuration of newly fabricated chips 001 Open Configuration - the configuration after Freescale- programmable fuses have been blown 01x Closed Configuration - the configuration after OEM-programmable fuses have been blown 1xx Field Return Configuration - the configuration of chips that are returned to Freescale for analysis
11–8 SSM_ST	System Security Monitor State  This field contains the encoded state of the SSM's state machine. The encoding of the possible states are:  0000 Init 1000 Init Intermediate (transition state between Init and Check - SSM stays in this state only one clock cycle) 1001 Check 1011 Non-Secure 1101 Trusted 1111 Secure 0011 Soft Fail 0001 Hard Fail
7–5 -	This field is reserved. Reserved
4 LPDIS	Low Power Disable  If 1, the SNVS low power section has been disabled by means of an input signal to the SNVS .
3–2 -	This field is reserved. Reserved
1 PI	Periodic Interrupt  Indicates that periodic interrupt has occurred since this bit was last cleared.  0 No periodic interrupt occurred. 1 A periodic interrupt occurred.
0 HPTA	HP Time Alarm Indicates that the HP Time Alarm has occurred since this bit was last cleared.  0 No time alarm interrupt occurred. 1 A time alarm interrupt occurred.

## 6.10.7 SNVS\_HP Security Violation Status Register (SNVS\_HPSVSR)

The HP Security Violation Status Register reflects the HP domain security violation records. Write a 1 to SEC\_VIO5-0 to clear the corresponding security violation detection flag. Note that this does not automatically clear the security violation signal that is connected to the input, so the security violation may immediately be detected again.

Address: 0h base + 18h offset = 18h



### SNVS\_HPSVSR field descriptions

Field	Description
31–28 -	This field is reserved. Reserved
27 ZMK_ECC_FAIL	Zeroizable Master Key Error Correcting Code Check Failure When set, this bit triggers a bad key violation to the SSM and a security violation to the SNVS_LP section, which clears security sensitive data. Writing a one to this bit clears the record of this failure. It also clears this register's ZMK_SYNDROME field.  0 ZMK ECC Failure was not detected. 1 ZMK ECC Failure was detected.

Table continues on the next page...

**SNVS\_HPSVSR field descriptions (continued)**

Field	Description
26–25 -	This field is reserved. Reserved
24–16 ZMK_ SYNDROME	Zeroizable Master Key Syndrome Value  The ZMK syndrome indicates error location and parity similar to the OTPMK syndrome . This value is set and locked when a ZMK ECC failure is detected. It is cleared by writing one into ZMK_ECC_FAIL bit.
15–6 -	This field is reserved. Reserved
5 SEC_VIO5	Security violation on input 5 is detected.  0 No security violation occurred on port 5. 1 Security violation occurred on port 5.
4 SEC_VIO4	Security violation on input 4 is detected.  0 No security violation occurred on port 4. 1 Security violation occurred on port 4.
3 SEC_VIO3	Security violation on input 3 is detected.  0 No security violation occurred on port 3. 1 Security violation occurred on port 3.
2 SEC_VIO2	Security violation on input 2 is detected.  0 No security violation occurred on port 2. 1 Security violation occurred on port 2.
1 SEC_VIO1	Security violation on input 1 is detected.  0 No security violation occurred on port 1. 1 Security violation occurred on port 1.
0 SEC_VIO0	Security violation on input 0 is detected.  0 No security violation occurred on port 0. 1 Security violation occurred on port 0.

### 6.10.8 SNVS\_HP High Assurance Counter IV Register (SNVS\_HPHACIVR)

The SNVS \_HP High Assurance Counter IV Register contains the initial value for the high assurance counter.

Address: 0h base + 1Ch offset = 1Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_HPHACIVR field descriptions**

Field	Description
31–0 HAC_ COUNTER_IV	High Assurance Counter Initial Value  This register is used to set the starting count value to the high assurance counter. This register cannot be programmed when HAC_L bit is set.

**6.10.9 SNVS\_HP High Assurance Counter Register (SNVS\_HPHACR)**

The SNVS \_HP High Assurance Counter Register contains the value of the high assurance counter. The high assurance counter is a delay introduced before the system security monitor transitions from soft fail to hard fail state if this transition is enabled.

Address: 0h base + 20h offset = 20h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_HPHACR field descriptions**

Field	Description
31–0 HAC_COUNTER	High Assurance Counter  When the HAC_EN bit is set and the SSM is in the soft fail state, this counter starts to count down with the system clock. When the counter reaches zero, the SSM transitions to the Hard Fail State. <ul style="list-style-type: none"><li>• When HAC_STOP bit is set, the HAC Counter is stopped.</li><li>• When HAC_CLEAR bit is set, the HAC Counter is cleared.</li><li>• When HAC_LOAD bit is set, the HAC Counter is loaded with the value of the HPHACIVR.</li></ul>

**6.10.10 SNVS\_HP Real Time Counter MSB Register (SNVS\_HPRTCMR)**

The SNVS \_HP Real Time Counter MSB register contains the most significant bits of the HP Real Time Counter.

Address: 0h base + 24h offset = 24h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_HPRTC MR field descriptions**

Field	Description
31–15 -	This field is reserved. Reserved
14–0 RTC	HP Real Time Counter Most significant 15 bits. This register can be programmed only when RTC is not active (RTC_EN bit is not set).

**6.10.11 SNVS\_HP Real Time Counter LSB Register (SNVS\_HPRTCLR)**

The SNVS \_HP Real Time Counter LSB register contains the 32 least significant bits of the HP real time counter.

Address: 0h base + 28h offset = 28h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_HPRTCLR field descriptions**

Field	Description
31–0 RTC	HP Real Time Counter Least significant 32 bits. This register can be programmed only when RTC is not active (RTC_EN bit is not set).

**6.10.12 SNVS\_HP Time Alarm MSB Register (SNVS\_HPTAMR)**

The SNVS \_HP Time Alarm MSB register contains the most significant bits of the SNVS \_HP Time Alarm value.

Address: 0h base + 2Ch offset = 2Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_HPTAMR field descriptions**

Field	Description
31–15 -	This field is reserved. Reserved
14–0 HPTA	HP Time Alarm Most significant 15 bits. This register can be programmed only when HP time alarm is disabled (HPTA_EN bit is not set).

**6.10.13 SNVS\_HP Time Alarm LSB Register (SNVS\_HPTALR)**

The SNVS \_HP Time Alarm LSB register contains the 32 least significant bits of the SNVS \_HP Time Alarm value.

Address: 0h base + 30h offset = 30h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0

**SNVS\_HPTALR field descriptions**

Field	Description
31–0 HPTA	HP Time Alarm Least significant bits. This register can be programmed only when HP time alarm is disabled (HPTA_EN bit is not set).

### 6.10.14 SNVS\_LP Lock Register (SNVS\_LPLR)

The SNVS\_LP Lock Register contains lock bits for the SNVS\_LP registers.

Address: 0h base + 34h offset = 34h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	Reserved						MKS_HL	LPTDCR_HL	LPTGFCR_HL	LPSVCR_HL	GPR_HL	MC_HL	LPCALB_HL	SRTC_HL	ZMK_RHL	ZMK_WHL
W	Reserved						0	0	0	0	0	0	0	0	0	0

#### SNVS\_LPLR field descriptions

Field	Description
31–10 -	This field is reserved. Reserved
9 MKS_HL	Master Key Select Hard Lock When set, prevents any writes to the MASTER_KEY_SEL field of the LP Master Key Control Register. Once set, this bit can only be reset by the LP POR.  0 Write access is allowed. 1 Write access is not allowed.
8 LPTDCR_HL	LP Tamper Detectors Configuration Register Hard Lock When set, prevents any writes to the LPTDCR. Once set, this bit can only be reset by the LP POR.  0 Write access is allowed. 1 Write access is not allowed.
7 LPTGFCR_HL	LP Tamper Glitch Filter Configuration Register Hard Lock When set, prevents any writes to the LPTGFCR. Once set, this bit can only be reset by the LP POR.  0 Write access is allowed. 1 Write access is not allowed.
6 LPSVCR_HL	LP Security Violation Control Register Hard Lock When set, prevents any writes to the LPSVCR. Once set, this bit can only be reset by the LP POR.

Table continues on the next page...

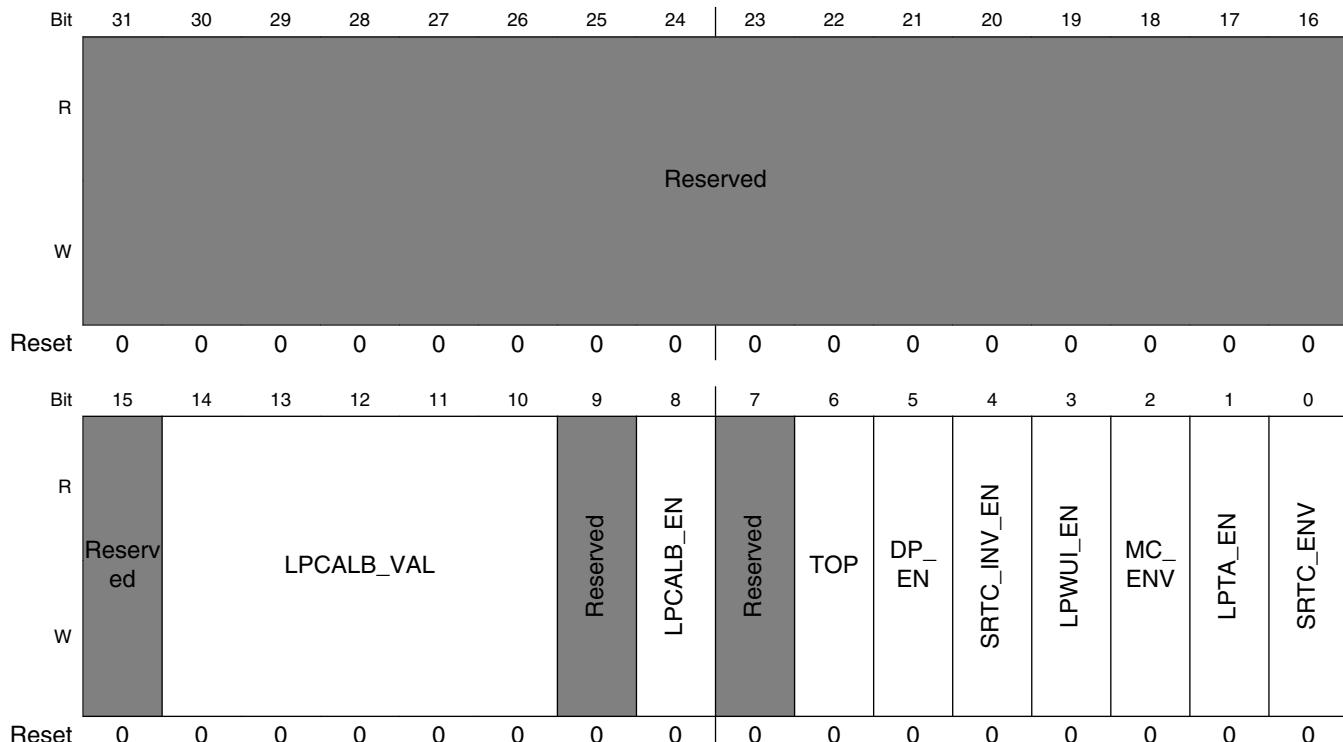
**SNVS\_LPLR field descriptions (continued)**

Field	Description
	<p>0 Write access is allowed. 1 Write access is not allowed.</p>
5 GPR_HL	<p>General Purpose Register Hard Lock When set, prevents any writes to the GPR. Once set, this bit can only be reset by the LP POR.</p> <p>0 Write access is allowed. 1 Write access is not allowed.</p>
4 MC_HL	<p>Monotonic Counter Hard Lock When set, prevents any writes (increments) to the MC Registers and MC_ENV bit. Once set, this bit can only be reset by the LP POR.</p> <p>0 Write access (increment) is allowed. 1 Write access (increment) is not allowed.</p>
3 LPCALB_HL	<p>LP Calibration Hard Lock When set, prevents any writes to the LP Calibration Value (LPCALB_VAL) and LP Calibration Enable (LPCALB_EN). Once set, this bit can only be reset by the LP POR.</p> <p>0 Write access is allowed. 1 Write access is not allowed.</p>
2 SRTC_HL	<p>Secure Real Time Counter Hard Lock When set, prevents any writes to the SRTC registers, SRTC_ENV, and SRTC_INV_EN bits. Once set, this bit can only be reset by the LP POR.</p> <p>0 Write access is allowed. 1 Write access is not allowed.</p>
1 ZMK_RHL	<p>Zeroizable Master Key Read Hard Lock When set, prevents any software reads to the ZMK registers and ZMK_ECC_VALUE field of the LPMKCR. In ZMK hardware programming mode (ZMK_HWP is set), software cannot read the ZMK or ZMK_ECC_VALUE. Regardless of the setting of this bit, hardware can use the ZMK value when ZMK is selected. Once set, this bit can only be reset by the LP POR.</p> <p>0 Read access is allowed (only in software programming mode). 1 Read access is not allowed.</p>
0 ZMK_WHL	<p>Zeroizable Master Key Write Hard Lock When set, prevents any writes (software and hardware) to the ZMK registers and ZMK_HWP, ZMK_VAL, and ZMK_ECC_EN fields of the LPMKCR. Once set, this bit can only be reset by the LP POR.</p> <p>0 Write access is allowed. 1 Write access is not allowed.</p>

### 6.10.15 SNVS\_LP Control Register (SNVS\_LPCR)

The SNVS \_LP Control Register contains various control bits of the LP section of SNVS .

Address: 0h base + 38h offset = 38h



**SNVS\_LPCR field descriptions**

Field	Description												
31–15 -	This field is reserved. Reserved												
14–10 LPCALB_VAL	<p>LP Calibration Value</p> <p>Defines signed calibration value for SRTC. This field can be programmed only when SRTC calibration is disabled and not locked, i.e. when LPCALB_EN, LPCALB_SL, and LPCALB_HL bits are not set. This is a 5-bit 2's complement value. Hence, the allowable calibration values are in the range from -16 to +15 counts per 32768 ticks of the counter clock</p> <table> <tbody> <tr> <td>00000</td> <td>+0 counts per each 32768 ticks of the counter clock</td> </tr> <tr> <td>00001</td> <td>+1 counts per each 32768 ticks of the counter clock</td> </tr> <tr> <td>00010</td> <td>+2 counts per each 32768 ticks of the counter clock</td> </tr> <tr> <td>01111</td> <td>+15 counts per each 32768 ticks of the counter clock</td> </tr> <tr> <td>10000</td> <td>-16 counts per each 32768 ticks of the counter clock</td> </tr> <tr> <td>10001</td> <td>-15 counts per each 32768 ticks of the counter clock</td> </tr> </tbody> </table>	00000	+0 counts per each 32768 ticks of the counter clock	00001	+1 counts per each 32768 ticks of the counter clock	00010	+2 counts per each 32768 ticks of the counter clock	01111	+15 counts per each 32768 ticks of the counter clock	10000	-16 counts per each 32768 ticks of the counter clock	10001	-15 counts per each 32768 ticks of the counter clock
00000	+0 counts per each 32768 ticks of the counter clock												
00001	+1 counts per each 32768 ticks of the counter clock												
00010	+2 counts per each 32768 ticks of the counter clock												
01111	+15 counts per each 32768 ticks of the counter clock												
10000	-16 counts per each 32768 ticks of the counter clock												
10001	-15 counts per each 32768 ticks of the counter clock												

*Table continues on the next page...*

**SNVS\_LPCR field descriptions (continued)**

Field	Description
	11110 -2 counts per each 32768 ticks of the counter clock 11111 -1 counts per each 32768 ticks of the counter clock
9 -	This field is reserved. Reserved
8 LPCALB_EN	LP Calibration Enable When set, enables the SRTC calibration mechanism. This bit cannot be changed once LPCALB_SL or LPCALB_HL bit is set.  0 SRTC Time calibration is disabled. 1 SRTC Time calibration is enabled.
7 -	This field is reserved. Reserved
6 TOP	Turn off System Power Asserting this bit causes a signal to be sent to the Power Management IC to turn off the system power. This bit will clear once power is off. This bit is only valid when the Dumb PMIC is enabled.  0 Leave system power on. 1 Turn off system power.
5 DP_EN	Dumb PMIC Enabled When set, software can control the system power. When cleared, the system requires a Smart PMIC to automatically turn power off.  0 Smart PMIC enabled. 1 Dumb PMIC enabled.
4 SRTC_INV_EN	Secure Real Time Counter Invalidation Enable When set, the SRTC is invalidated (SRTC_ENV bit is cleared) in the case of security violation. This field cannot be changed once SRTC_SL or SRTC_HL bit is set.  0 SRTC stays valid in the case of security violation. 1 SRTC is invalidated in the case of security violation.
3 LPWUI_EN	LP Wake-Up Interrupt Enable This interrupt line should be connected to the external pin and is intended to inform the external chip about an SNVS_LP event (tamper event, MC rollover, SRTC rollover, or time alarm). This wake-up signal can be asserted only when the chip (HP section) is powered down, and the LP section is isolated.  0 LP wake-up interrupt is disabled. 1 LP wake-up interrupt is enabled.
2 MC_ENV	Monotonic Counter Enable and Valid When set, the MC can be incremented (by write transaction to the LPSMCMR or LPSMCLR). This bit cannot be changed once MC_SL or MC_HL bit is set.  0 MC is disabled or invalid. 1 MC is enabled and valid.
1 LPTA_EN	LP Time Alarm Enable When set, the SNVS functional interrupt is asserted if the LP Time Alarm Register is equal to the 32 MSBs of the secure real time counter.

*Table continues on the next page...*

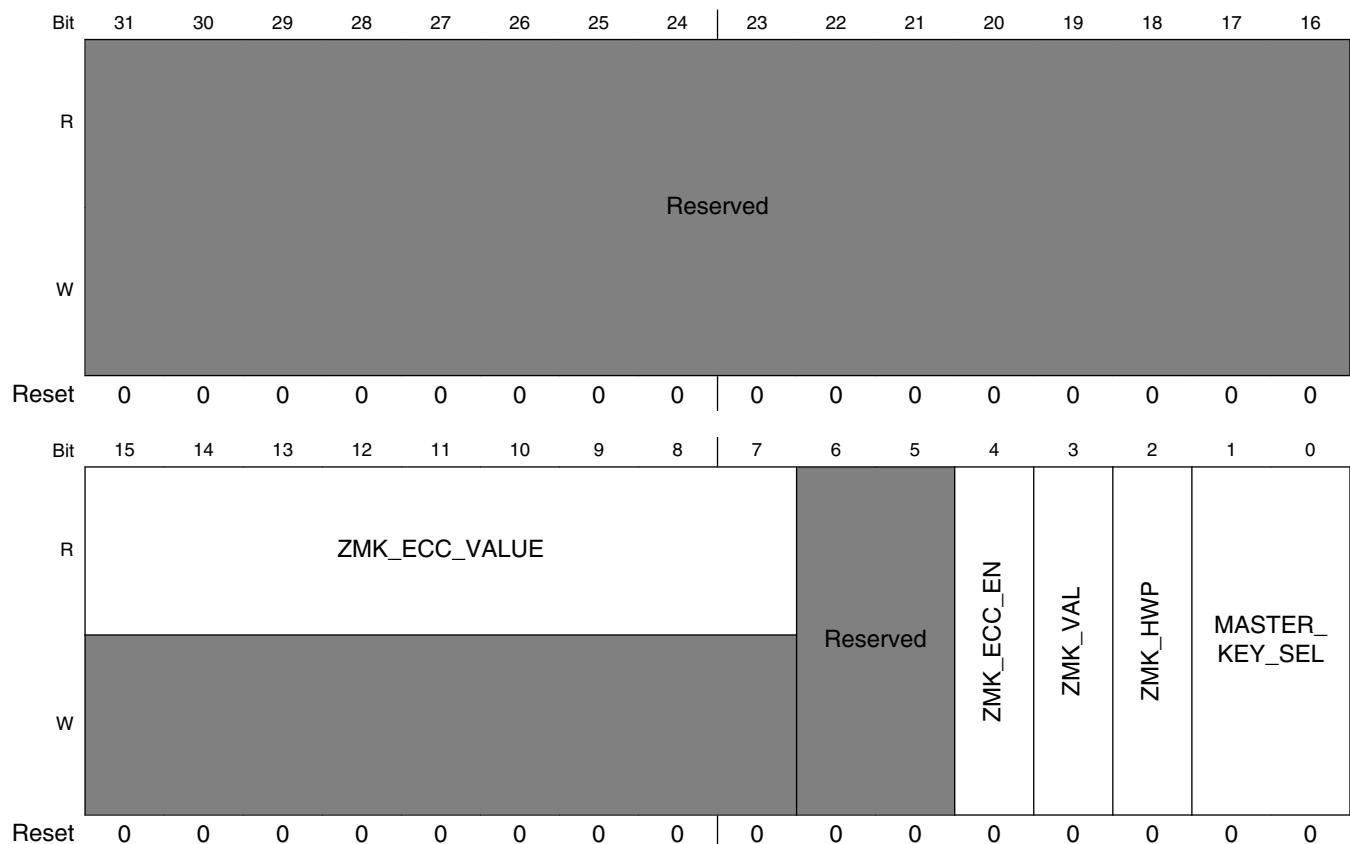
**SNVS\_LPCR field descriptions (continued)**

Field	Description
	0 LP time alarm interrupt is disabled. 1 LP time alarm interrupt is enabled.
0 SRTC_ENV	Secure Real Time Counter Enable and Valid When set, the SRTC becomes operational. This bit cannot be changed once SRTC_SL or SRTC_HL bit is set. 0 SRTC is disabled or invalid. 1 SRTC is enabled and valid.

**6.10.16 SNVS\_LP Master Key Control Register (SNVS\_LPMKCR)**

The SNVS\_LP Master Key Control Register contains the master keys configuration.

Address: 0h base + 3Ch offset = 3Ch



**SNVS\_LPMKCR field descriptions**

Field	Description
31–16 -	This field is reserved. Reserved
15–7 ZMK_ECC_ VALUE	Zeroizable Master Key Error Correcting Code Value  This field is automatically calculated and set when one is written into ZMK_ECC_EN bit of this register. This field cannot be programmed by software. It keeps the ECC value of the zeroizable master key, which allows checking that ZMK has not been corrupted/altered with time.  Note that this ZMK ECC code is equivalent to the ECC bits encoded into the OTPMK value but for the ZMK, the ECC value is kept separate from the ZMK value. See <a href="#">Error code for the ZMK</a> for details. Read restrictions similar to the ZMK Registers are applied to this field (see <a href="#">SNVS_LP Zeroizable Master Key Register n (SNVS_LPZMKRn)</a> ).
6–5 -	This field is reserved. Reserved
4 ZMK_ECC_EN	Zeroizable Master Key Error Correcting Code Check Enable  Writing one to this field automatically calculates and sets the ZMK ECC value in the ZMK_ECC_VALUE field of this register. When both ZMK value is valid (ZMK_VAL is set) and ZMK ECC check is enabled (ZMK_ECC_EN is set), the ZMK value is continuously checked for the valid ECC word. If the ZMK ECC word calculated every clock cycle does not match the one recorded in this register, the ZMK ECC Check Fail Violation is generated. This bit cannot be programmed when ZMK_WSL or ZMK_WHL bit is set.  0 ZMK ECC check is disabled. 1 ZMK ECC check is enabled.
3 ZMK_VAL	Zeroizable Master Key Valid  When set, the ZMK value can be selected by the master key control block for use by cryptographic modules. In hardware programming mode, hardware sets this bit when the ZMK provisioning is complete. In software programming mode, software should set this bit. This bit cannot be programmed when ZMK_WSL or ZMK_WHL bit is set.  0 ZMK is not valid. 1 ZMK is valid.
2 ZMK_HWP	Zeroizable Master Key hardware Programming mode  When set, only the hardware key programming mechanism can set the ZMK and software cannot read it. When not set, the ZMK can be programmed only by software. See <a href="#">ZMK hardware programming mechanism</a> for details. This bit cannot be programmed when ZMK_WSL or ZMK_WHL bit is set.  0 ZMK is in the software programming mode. 1 ZMK is in the hardware programming mode.
1–0 MASTER_KEY_ SEL	Master Key Select  These bits select the SNVS Master Key output when Master Key Select bits are enabled by MKS_EN bit in the HPCOMR . When MKS_EN bit is not set, the one time programmable master key is selected by default. This field cannot be programmed when MKS_SL(or the hard lock) bit is set.  0x Select one time programmable master key. 10 Select zeroizable master key when MKS_EN bit is set . 11 Select combined master key when MKS_EN bit is set .

### 6.10.17 SNVS\_LP Security Violation Control Register (SNVS\_LPSVCR)

The LP Security Violation Control Register configures security violation inputs. This register cannot be programmed when the LPSVCR Lock bit is set. Note that configurations of the security violation inputs in the HP section (HPSVCR) and LP section (LPSVCR Register) are independent and have different functionality.

Address: 0h base + 40h offset = 40h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

#### SNVS\_LPSVCR field descriptions

Field	Description
31–6 -	This field is reserved. Reserved
5 SV_EN5	Security Violation 5 Enable  This bit enables security violation input 5. When set, a security violation 5 causes an LP security violation, which clears LP sensitive data.  0 Security Violation 5 is disabled in the LP domain. 1 Security Violation 5 is enabled in the LP domain.
4 SV_EN4	Security Violation 4 Enable  This bit enables security violation input 4. When set, a security violation 4 causes an LP security violation, which clears LP sensitive data.  0 Security Violation 4 is disabled in the LP domain. 1 Security Violation 4 is enabled in the LP domain.
3 SV_EN3	Security Violation 3 Enable  This bit enables security violation input 3. When set, a security violation 3 causes an LP security violation, which clears LP sensitive data.

Table continues on the next page...

**SNVS\_LPSVCR field descriptions (continued)**

Field	Description
	<p>0 Security Violation 3 is disabled in the LP domain. 1 Security Violation 3 is enabled in the LP domain.</p>
2 SV_EN2	<p><b>Security Violation 2 Enable</b> This bit enables security violation input 2. When set, a security violation 2 causes an LP security violation, which clears LP sensitive data.</p> <p>0 Security Violation 2 is disabled in the LP domain. 1 Security Violation 2 is enabled in the LP domain.</p>
1 SV_EN1	<p><b>Security Violation 1 Enable</b> This bit enables security violation input 1. When set, a security violation 1 causes an LP security violation, which clears LP sensitive data.</p> <p>0 Security Violation 1 is disabled in the LP domain. 1 Security Violation 1 is enabled in the LP domain.</p>
0 SV_EN0	<p><b>Security Violation 0 Enable</b> This bit enables security violation input 0. When set, a security violation 0 causes an LP security violation, which clears LP sensitive data.</p> <p>0 Security Violation 0 is disabled in the LP domain. 1 Security Violation 0 is enabled in the LP domain.</p>

## 6.10.18 SNVS\_LP Tamper Glitch Filters Configuration Register (SNVS\_LPTGFCR)

The SNVS \_LP Tamper Glitch Filters Configuration Register is used to configure the glitch filters for the SNVS \_LP tamper inputs. This register cannot be programmed when the LPTGFCR\_SL or LPTGFCR\_HL bit is set.

Address: 0h base + 44h offset = 44h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ETGF2_EN	Reserved			ETGF2				ETGF1_EN	Reserved			ETGF1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								WMTGF_EN	Reserved			WMTGF			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### SNVS\_LPTGFCR field descriptions

Field	Description
31 ETGF2_EN	External Tamper Glitch Filter 2 Enable When set, enables the external tamper glitch filter 2. 0 External tamper glitch filter 2 is bypassed. 1 External tamper glitch filter 2 is enabled.
30–29 -	This field is reserved. Reserved
28–24 ETGF2	External Tamper Glitch Filter 2 Configures the length of the digital glitch filter for external tamper 2 pin between 128 and 8064 SRTC clock cycles. Any assertion on external tamper 2 that is equal to or less than the value of the digital glitch filter is ignored. The length of the glitches filtered out is: $128 + (\text{ETGF2} \times 256)$ , where ETGF2 = 0, ..., 31
23 ETGF1_EN	External Tamper Glitch Filter 1 Enable When set, enables the external tamper glitch filter 1.

Table continues on the next page...

**SNVS\_LPTGFCR field descriptions (continued)**

Field	Description
	<p>0 External tamper glitch filter 1 is bypassed. 1 External tamper glitch filter 1 is enabled.</p>
22–21 -	This field is reserved. Reserved
20–16 ETGF1	<p>External Tamper Glitch Filter 1</p> <p>Configures the length of the digital glitch filter for the external tamper 1 pin between 128 and 8064 SRTC clock cycles. Any assertion on external tamper 1 that is equal to or less than the value of the digital glitch filter is ignored.</p> <p>The length of the glitches filtered out is:</p> $128 + (\text{ETGF1} \times 256), \text{ where ETGF1} = 0, \dots, 31$
15–8 -	This field is reserved. Reserved
7 WMTGF_EN	<p>Wire-Mesh Tamper Glitch Filter Enable</p> <p>When set, enables the wire-mesh tamper glitch filter. Note that two wire-mesh tamper inputs (1 and 2) share this glitch filter.</p> <p>0 Wire-mesh tamper glitch filter is bypassed. 1 Wire-mesh tamper glitch filter is enabled.</p>
6–5 -	This field is reserved. Reserved
4–0 WMTGF	<p>Wire-Mesh Tamper Glitch Filter</p> <p>Configures the length of the digital glitch filter for the wire-mesh tamper 1 and 2 pins between 1 and 63 SRTC clock cycles. Two wire-mesh tamper inputs share this glitch filter. Any assertion on wire-mesh tamper 1 or 2 that is equal to or less than the value of the digital glitch filter is ignored.</p> <p>The length of the glitches filtered out is:</p> $1 + (\text{WMTGF} \times 2), \text{ where WMTGF} = 0, \dots, 31$

### 6.10.19 SNVS\_LP Tamper Detectors Configuration Register (SNVS\_LPTDCR)

The SNVS\_LP Tamper Detectors Configuration Register is used to configure analog and digital tamper detector sources. This register cannot be programmed when LPTDCR is locked for write.

Address: 0h base + 48h offset = 48h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
	Reserved			OSCB	Reserved		VRC		Reserved		HTDC		Reserved		LTDC	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	POR_OBSERV	PFD_OBSERV			Reserved	ET2_EN	ET1_EN	WMT2_EN	WMT1_EN	VT_EN	TT_EN	CT_EN	Reserved	MCR_EN	SRTC_EN	Reserved
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SNVS\_LPTDCR field descriptions**

Field	Description
31–29 -	This field is reserved. Reserved
28 OSCB	Oscillator Bypass  When set, this signal bypasses the internal SRTC clock oscillator, allowing only an external clock to drive the SRTC clock source. This bit is an output of the SNVS ; the oscillator bypass mechanism should be implemented at the chip level (outside of the SNVS ) if required.  0 Oscillator not bypassed. 1 Oscillator bypassed. External clock can drive the SRTC clock source.
27 -	This field is reserved. Reserved

*Table continues on the next page...*

**SNVS\_LPTDCR field descriptions (continued)**

Field	Description
26–24 VRC	Voltage Reference Configuration  These configuration bits are wired as an output of the module.
23 -	This field is reserved. Reserved
22–20 HTDC	High Temperature Detect Configuration  These configuration bits are wired as an output of the module.
19 -	This field is reserved. Reserved
18–16 LTDC	Low Temperature Detect Configuration  These configuration bits are wired as an output of the module.
15 POR_OBSERV	Power On Reset (POR) Observability Flop  The asynchronous reset input of this flop is connected directly to the output of the POR analog circuitry (external to the SNVS block). This flop can be used to detect brown-out voltage of the POR circuitry.
14 PFD_OBSERV	System Power Fail Detector (PFD) Observability Flop  The asynchronous reset input of this flop is connected directly to the inverted output of the PFD analog circuitry (external to the SNVS block). This flop can be used to detect brown-out voltage of the PFD circuitry.
13–11 -	This field is reserved. Reserved
10 ET2_EN	External Tampering 2 Enable  When set, external tampering 2 detection generates an LP security violation.  0 External tamper 2 is disabled. 1 External tamper 2 is enabled.
9 ET1_EN	External Tampering 1 Enable  When set, external tampering 1 detection generates an LP security violation.  0 External tamper 1 is disabled. 1 External tamper 1 is enabled.
8 WMT2_EN	Wire-Mesh Tampering 2 Enable  When set, wire-mesh tampering 2 detection generates an LP security violation.  0 Wire-mesh tamper 2 is disabled. 1 Wire-mesh tamper 2 is enabled.
7 WMT1_EN	Wire-Mesh Tampering 1 Enable  When set, wire-mesh tampering 1 detection generates an LP security violation.  0 Wire-mesh tamper 1 is disabled. 1 Wire-mesh tamper 1 is enabled.
6 VT_EN	Voltage Tamper Enable  When set, a voltage monitor tamper generates an LP security violation.  0 Voltage tamper is disabled. 1 Voltage tamper is enabled.

*Table continues on the next page...*

**SNVS\_LPTDCR field descriptions (continued)**

Field	Description
5 TT_EN	Temperature Tamper Enable When set, a temperature monitor tamper generates an LP security violation.  0 Temperature tamper is disabled. 1 Temperature tamper is enabled.
4 CT_EN	Clock Tamper Enable When set, a clock monitor tamper generates an LP security violation.  0 Clock tamper is disabled. 1 Clock tamper is enabled.
3 -	This field is reserved. Reserved
2 MCR_EN	MC Rollover Enable When set, an MC Rollover event generates an LP security violation.  0 MC rollover is disabled. 1 MC rollover is enabled.
1 SRTC_R_EN	SRTC Rollover Enable When set, an SRTC rollover event generates an LP security violation.  0 SRTC rollover is disabled. 1 SRTC rollover is enabled.
0 -	This field is reserved. Reserved

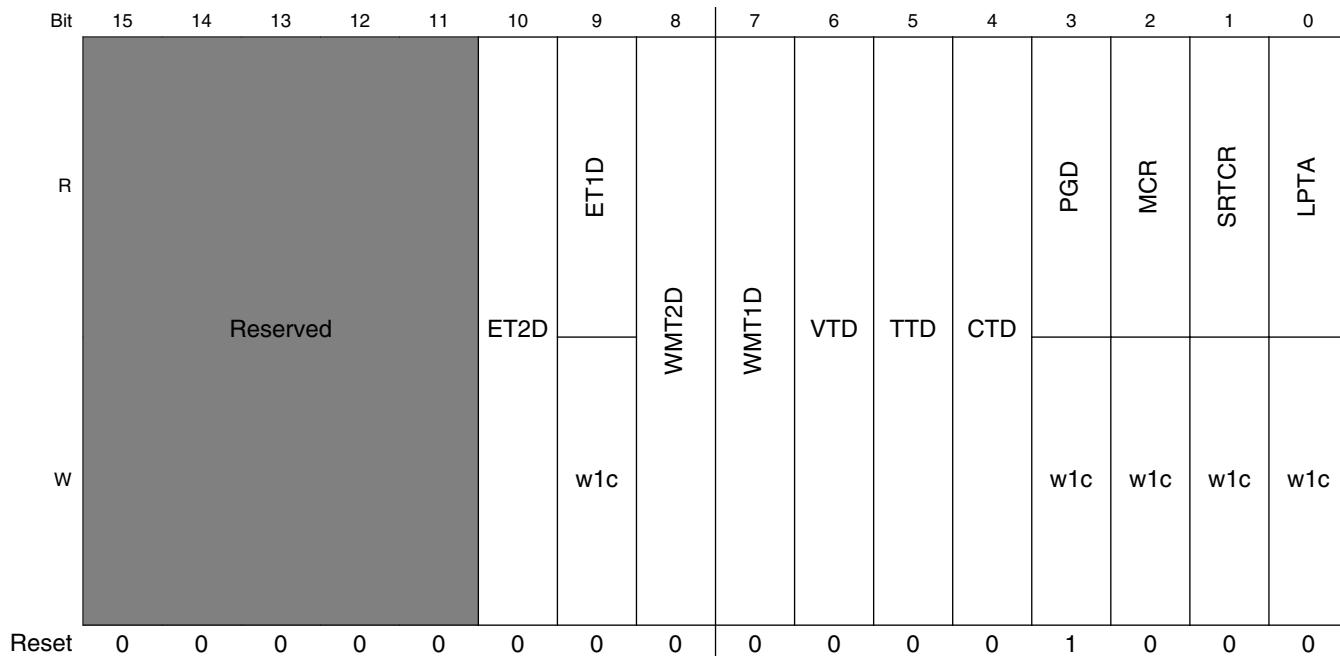
### 6.10.20 SNVS\_LP Status Register (SNVS\_LPSR)

The SNVS \_LP Status Register reflects the internal state and behavior of the SNVS \_LP.

Address: 0h base + 4Ch offset = 4Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LPS	LPNS										SED		SPO	EO	ESVD
W												w1c		w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

## SNVS Memory Map/Register Definition



### SNVS\_LPSR field descriptions

Field	Description
31 LPS	<p>LP Section is Secured</p> <p>Indicates that the LP section is provisioned/programmed in the secure or trusted state. The first write to the LP registers in secure or trusted state sets this bit. This bit can never be set together with the LPNS bit. When set the SNVS_LP section cannot be programmed , and ZMK cannot be read in the non-secure state of the SSM .</p> <p>0 LP section was not programmed in secure or trusted state. 1 LP section was programmed in secure or trusted state.</p>
30 LPNS	<p>LP Section is Non-Secured</p> <p>Indicates that LP section was provisioned/programmed in the non-secure state. The first successful write to the LP Registers in non-secure state sets this bit. This bit can never be set together with the LPS bit. When set, the entire SNVS_LP section (all LP registers) are cleared upon an SSM transition from check to trusted state.</p> <p>0 LP section was not programmed in the non-secure state. 1 LP section was programmed in the non-secure state.</p>
29–21 -	This field is reserved. Reserved
20 SED	<p>Scan Exit Detected</p> <p>0 Scan exit was not detected. 1 Scan exit was detected.</p>
19 -	This field is reserved. Reserved
18 SPO	<p>Set Power Off</p> <p>This bit is set when power off was requested by a button press, but the button was not pressed for 5 seconds.</p>

Table continues on the next page...

**SNVS\_LPSR field descriptions (continued)**

Field	Description
	0 Emergency Off was not detected. 1 Emergency Off was detected..
17 EO	Emergency Off This bit is set when a power off is requested. 0 Emergency off was not detected. 1 Emergency off was detected.
16 ESVD	External Security Violation Detected Indicates that a security violation is detected on one of the HP security violation ports. The record of the port on which the violation has occurred can be found in the HP Security Violation Status Register. 0 No external security violation. 1 External security violation is detected.
15–11 -	This field is reserved. Reserved
10 ET2D	External Tampering 2 Detected. 0 External tampering 2 not detected. 1 External tampering 2 detected.
9 ET1D	External Tampering 1 Detected. 0 External tampering 1 not detected. 1 External tampering 1 detected.
8 WMT2D	Wire-Mesh Tampering 2 Detected. 0 Wire-mesh tampering 2 not detected. 1 Wire-mesh tampering 2 detected.
7 WMT1D	Wire-Mesh Tampering 1 Detected. 0 Wire-mesh tampering 1 not detected. 1 Wire-mesh tampering 1 detected.
6 VTD	Voltage Tampering Detected. 0 Voltage tampering not detected. 1 Voltage tampering detected.
5 TTD	Temperature Tamper Detected. 0 No temperature tamper. 1 Temperature tamper is detected.
4 CTD	Clock Tampering Detected. 0 No clock tamper. 1 Clock tamper is detected.
3 PGD	Power Supply Glitch Detected. 0 No power supply glitch. 1 Power supply glitch is detected.

Table continues on the next page...

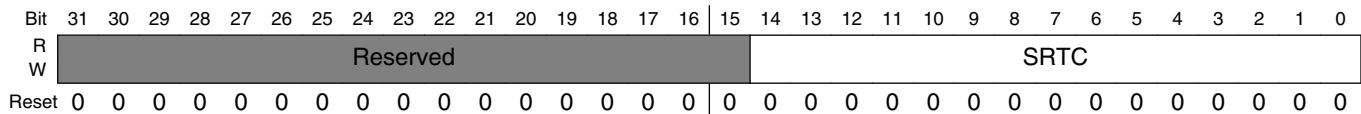
**SNVS\_LPSR field descriptions (continued)**

Field	Description
2 MCR	Monotonic Counter Rollover. 0 MC has not reached its maximum value. 1 MC has reached its maximum value.
1 SRTCR	Secure Real Time Counter Rollover. 0 SRTC has not reached its maximum value. 1 SRTC has reached its maximum value.
0 LPTA	LP Time Alarm. 0 No time alarm interrupt occurred. 1 A time alarm interrupt occurred.

### 6.10.21 SNVS\_LP Secure Real Time Counter MSB Register (SNVS\_LPSRTCMR)

The SNVS\_LP Secure Real Time Counter MSB register contains the most-significant bits of the secure real time counter.

Address: 0h base + 50h offset = 50h

**SNVS\_LPSRTCMR field descriptions**

Field	Description
31–15 -	This field is reserved. Reserved
14–0 SRTC	LP Secure Real Time Counter most significant 15 bits This register can be programmed only when SRTC is not active and not locked, meaning the SRTC_ENV, SRTC_SL, and SRTC_HL bits are not set.

### 6.10.22 SNVS\_LP Secure Real Time Counter LSB Register (SNVS\_LPSRTCLR)

The SNVS \_LP Secure Real Time Counter LSB register contains the 32 least-significant bits of the secure real time counter.

Address: 0h base + 54h offset = 54h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																																	
W																																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

#### SNVS\_LPSRTCLR field descriptions

Field	Description
31–0 SRTC	LP Secure Real Time Counter least significant 32 bits  This register can be programmed only when SRTC is not active and not locked, meaning the SRTC_ENV, SRTC_SL, and SRTC_HL bits are not set.

### 6.10.23 SNVS\_LP Time Alarm Register (SNVS\_LPTAR)

The SNVS \_LP Time Alarm register contains the 32-bit LP Time Alarm value.

Address: 0h base + 58h offset = 58h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

#### SNVS\_LPTAR field descriptions

Field	Description
31–0 LPTA	LP Time Alarm  This register can be programmed only when the LP time alarm is disabled (LPTA_EN bit is not set).

### 6.10.24 SNVS\_LP Secure Monotonic Counter MSB Register (SNVS\_LPSMCMR)

The SNVS \_LP Secure Monotonic Counter MSB Register contains the monotonic counter era bits and the most significant 16 bits of the monotonic counter. The monotonic counter is incremented by one if there is a write command to the LPSMCMR or LPSMCLR register.

Address: 0h base + 5Ch offset = 5Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																																	
W																																	

Reset 0

#### SNVS\_LPSMCMR field descriptions

Field	Description
31–16 MC_ERA_BITS	Monotonic Counter Era Bits These bits are inputs to the module and typically connect to fuses.
15–0 MON_COUNTER	Monotonic Counter Most Significant 16 Bits The MC is incremented by one when: <ul style="list-style-type: none"> <li>• A write transaction to the LPSMCMR or LPSMCLR register is detected.</li> <li>• The MC_ENV bit is set.</li> <li>• MC_SL and MC_HL bits are not set.</li> </ul>

### 6.10.25 SNVS\_LP Secure Monotonic Counter LSB Register (SNVS\_LPSMCLR)

The SNVS \_LP Secure Monotonic Counter LSB Register contains the 32 least significant bits of the monotonic counter. The MC is incremented by one if there is a write command to the LPSMCMR or LPSMCLR register.

Address: 0h base + 60h offset = 60h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																																	
W																																	

Reset 0

**SNVS\_LPSMCLR field descriptions**

Field	Description
31–0 MON_COUNTER	Monotonic Counter bits  The MC is incremented by one when: <ul style="list-style-type: none"> <li>• A write transaction to the LPSMCMR or LPSMCLR Register is detected.</li> <li>• The MC_ENV bit is set.</li> <li>• MC_SL and MC_HL bits are not set.</li> </ul>

**6.10.26 SNVS\_LP Power Glitch Detector Register (SNVS\_LPPGDR)**

The SNVS\_LP Power Glitch Detector Register provides a 32 bit read write register, which is used for storing power glitch detector value as described in [Power glitch detector \(PGD\)](#).

Address: 0h base + 64h offset = 64h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_LPPGDR field descriptions**

Field	Description
31–0 PGD	Power Glitch Detector Value

**6.10.27 SNVS\_LP General Purpose Register (SNVS\_LPGPR)**

The SNVS\_LP General Purpose Register provides a 32 bit read write register, which can be used by any application for retaining 32 bit data during a power-down mode.

Address: 0h base + 68h offset = 68h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W																																

Reset 0

**SNVS\_LPGPR field descriptions**

Field	Description
31–0 GPR	General Purpose Register When GPR_SL or GPR_HL bit is set, the register cannot be programmed.

**6.10.28 SNVS\_LP Zeroizable Master Key Register n  
(SNVS\_LPZMKRn)**

The SNVS\_LP Zeroizable Master Key Registers contain the 256-bit zeroizable master key value. These registers are programmable as follows:

- When ZMK write lock bit is set, they cannot be programmed.
- When ZMK\_HWP is not set, they are in software programming mode and can be programmed only by software.
- When ZMK\_HWP is set, they are in hardware programming mode and can be programmed only by hardware.

These registers cannot be read by software when the ZMK\_HWP or ZMK read lock bit is set.

Address: 0h base + 6Ch offset + (4d × i), where i=0d to 7d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																	ZMK															
W																																

Reset 0

**SNVS\_LPZMKRn field descriptions**

Field	Description
31–0 ZMK	Zeroizable Master Key Each of these registers contains part of the 256-bit ZMK value: LPZMKR0 - ZMK[ 31-0 ] LPZMKR1 - ZMK[ 63-32 ] LPZMKR2 - ZMK[ 95-64 ] LPZMKR3 - ZMK[ 127-96 ] LPZMKR4 - ZMK[ 159-128 ] LPZMKR5 - ZMK[ 191-160 ] LPZMKR6 - ZMK[ 223-192 ] LPZMKR7 - ZMK[ 255-224 ]

### 6.10.29 SNVS\_HP Version ID Register 1 (SNVS\_HPV IDR1)

The SNVS \_HP Version ID Register 1 is a read-only register that contains the current version of the SNVS . The version consists of a module ID, a major version number, and a minor version number.

Address: 0h base + BF8h offset = BF8h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IP_ID																MAJOR_REV				MINOR_REV											
W																																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	

#### SNVS\_HPV IDR1 field descriptions

Field	Description
31–16 IP_ID	SNVS block ID
15–8 MAJOR_REV	SNVS block major version number
7–0 MINOR_REV	SNVS block minor version number

### 6.10.30 SNVS\_HP Version ID Register 2 (SNVS\_HPV IDR2)

The SNVS \_HP Version ID Register 2 is a read-only register that indicates the current version of the SNVS . The version consists of the following fields: integration options, ECO revision, and configuration options.

Address: 0h base + BFCh offset = BFCh

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved																INTG_OPT				ECO_REV				CONFIG_OPT							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

#### SNVS\_HPV IDR2 field descriptions

Field	Description
31–24 -	This field is reserved. Reserved

Table continues on the next page...

**SNVS\_HPVIDR2 field descriptions (continued)**

Field	Description
23–16 INTG_OPT	SNVS Integration Option
15–8 ECO_REV	SNVS ECO Revision
7–0 CONFIG_OPT	SNVS Configuration Option

# Appendix A

## Cryptographic Acceleration and Assurance Module (CAAM) Glossary

### A.1 Acronyms and abbreviations

Table A-1. Acronyms and abbreviated terms

Term	Meaning
AAD	Additional Authenticated Data
AES	Advanced Encryption Standard - 128-bit block encryption algorithm, using a 128, 192 or 256-bit key.
ARC4	Alleged RC4. Stream Cipher that is compatible with RC4.
AXI	AMBA Advanced eXtensible Interface (AXI) Protocol Specification. Defined by ARM Ltd.
CAAM	Cryptographic Acceleration and Assurance Module
CBC	Cipher Block Chaining An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
CCB	Cryptographic Control Block A logic module within CAAM
CCM	Counter with CBC-MAC Mode An authenticated encryption mode of operation. (Also known as CBC-MAC for CTR mode.)
CFB	Cipher FeedBack An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
CHA	Cryptographic Hardware Accelerator One of the hardware accelerators used in CAAM
CSP	Critical Security Parameter Security related information (such as secret and private cryptographic keys or authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. (See FIPS140-2)
CTR	Counter mode An encryption mode of operation used with AES
DECO	Descriptor Controller A logic module within CAAM
DEK	Data Encryption Key.

Table continues on the next page...

## Acronyms and abbreviations

**Table A-1. Acronyms and abbreviated terms (continued)**

Term	Meaning
DES	Data Encryption Standard 64-bit block encryption algorithm, using a 64-bit key.
3DES	Triple DES 64-bit block encryption algorithm, using a 128 or 196-bit key.
ECB	Electronic Code Book An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
GM	General Memory Memory that is not within CAAM's Secure Memory
HAB	High Assurance Boot software
HMAC	A hashing mode of operation used to implement a Message Authentication Code
IPAD	Inner padding defined for HMAC
ICV	Integrity Check Value A checksum or message digest that allows detection of errors or changes in data.
IV	Initialization Vector A value used to initialize some encryption modes of operation
KMOD	Key Modifier (field in SMAPJR register)
JDKEK	Job Descriptor Key Encryption Key
JQC	Job Queue Controller The hardware that schedules jobs received from the Job Rings
JR	Job Ring
MD5	A message digest algorithm returning a 128-bit hash value
MDHA	Message Digest Hardware Accelerator (hashing accelerator block)
MID	Master Identity Signals on the AXI bus that identify the bus master that initiated the transaction
MMU	Memory Management Unit
NS	Non-secure indication NS = 0 is secure. This signal is generated by the TrustZone feature implemented in some ARM processors. The Central Security Unit (CSU) generates this signal for other bus masters.
NVTK	Non-volatile Test Key
OFB	Output FeedBack An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
OPAD	Outer padding defined for HMAC
OTPMK	One-time-programmable Master Key
POR	Power On Reset.
PRNG	Pseudo Random Number Generator A deterministic algorithm that generates a sequence of numbers whose values are statistically random
PSP	Public Security Parameter Security-related public information whose modification can compromise the security of a cryptographic module.

*Table continues on the next page...*

**Table A-1. Acronyms and abbreviated terms (continued)**

Term	Meaning
RNG	Random Number Generator A hardware module within CAAM that generates random numbers based on the interaction of two free running ring oscillators and uses these random numbers to seed a PRNG.
SHA-1	A message digest algorithm defined in FIPS 180-2 returning a 160-bit hash value.
SHA-224	A message digest algorithm defined in FIPS 180-2 returning a 224-bit hash value.
SHA-256	A message digest algorithm defined in FIPS 180-2 returning a 256-bit hash value.
SHA-384	A message digest algorithm defined in FIPS 180-2 returning a 384-bit hash value.
SHA-512	A message digest algorithm defined in FIPS 180-2 returning a 512-bit hash value.
SMAG	Secure Memory Access Group (register)
SMAPJR	Secure Memory Access Permissions (register)
SM	Secure Memory A logic block within CAAM that provides access control and automatic zeroization for RAM
SNVS	Secure Non-Volatile Storage
SSP	Sensitive Security Parameter Data whose integrity must be protected
SWRST	Software Reset Register resets caused by writing 1 to the SWRST field in the MCFGR register.
TDSK	Trusted Descriptor Signing Key
TDKEK	Trusted Descriptor Key Encryption Key
TRK	Trusted Root Key
ZSK	Zeroizable Secret Key

## A.2 Glossary

**Table A-2. Glossary of terms**

Term	Description
Access control	The term 'access control' refers to intentional constraints placed on the ability of bus masters to read or write data, or execute instructions. Operating systems normally enforce access control over their processes' access to memory by adjusting the processor's MMU settings appropriately. But the processor's MMU cannot control the accesses made by other bus masters, including other processors. The access control features built into CAAM's Secure Memory are designed to control accesses from multiple bus masters. (See <i>Partition</i> . Also see <i>Own</i> .)
Alleged RC4	A stream cipher that is compatible with RC4.
Allocate	A processor allocates an unallocated partition to itself by writing into the partition's SMAPJR register. A processor allocates an unclaimed page of Secure Memory to itself by issuing an <i>Allocate Page</i> Command by means of the Secure Memory Command Register. (The phrase 'allocated to' is used interchangeably with 'claimed by'. Also see <i>Own</i> .)

Table continues on the next page...

**Table A-2. Glossary of terms (continued)**

Term	Description
Black blob	A blob whose input data when exporting and whose output when importing was assumed to be a black key. When exporting a black blob, the input data is first decrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB command was in a trusted descriptor) before being encrypted with the blob key. When importing a Black Blob, the data blob is first decrypted with the Blob Key before being encrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB Command was in a trusted descriptor). (See <i>Red Blob</i> .)
Black key	A key that has been encrypted using either the JDKEK or the TDKEK. (See <i>Red Key</i> .)
Blob	As used in this Block Guide the term 'blob' refers to a cryptographically protected data object consisting of a Blob Key encrypted with a Blob Key Encryption Key, a Data Blob encrypted with a Blob Key, and the MAC Tag resulting from the AES-CCM encryption of the Data Blob.
Blob key	The 256-bit random number used for AES-CCM encryption of the data portion of a blob.
Blob key encryption key	The Blob Key Encryption Key (BKEK) is a 256-bit key used when encrypting cryptographic Blobs exported from memory. It is intended for use in protecting the confidentiality and integrity of this data. The BKEK is derived from the Master Key or Non-volatile Test Key, a constant embedded in the CAAM Descriptor that initiated the Blob operation, the Security mode and the Blob type. (See <i>Master Key</i> .)
Claim	A processor claims ownership of an unallocated partition by writing into the partition's SMAPJR register. A processor claims an unclaimed page of Secure Memory by issuing an <i>Allocate Page</i> Command by means of Secure Memory Command Register. (The phrase "claimed by" is used interchangeably with "allocated to". See also <i>Own</i> .)
Critical security parameter	A critical security parameter (CSP) is security-related information (e.g., secret and private cryptographic keys, and authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)] A partition in Secure Memory can be marked as <i>CSP</i> , which causes CAAM to zeroize the pages in the partition in the event of a security violation, or if the partition is de-allocated. Individual pages that are de-allocated from a CSP partition are also zeroized.
Data encryption key	A data encryption key is a key that can be referenced in a descriptor as a cryptographic key and that is not one of other keys defined in this glossary. Some examples are: a symmetric key used for encryption or decryption of session data, an HMAC key.
De-allocate	A processor de-allocates a partition that it owns by issuing a De-allocate Partition Command via the Secure Memory Allocate Register. A partition marked PSP cannot be de-allocated. If the partition is marked CSP, the pages of that partition are zeroized before they are returned to the pool of unallocated pages. A processor de-allocates a page associated with a partition that the processor owns by issuing a <i>De-allocate Page</i> Command via the Secure Memory Allocate Register. A page associated with a PSP partition cannot be de-allocated. A page associated with a CSP partition is zeroized before it is returned to the pool of unallocated pages.
Decrypt key	A decrypt key is used for decrypting data to yield plaintext (unencrypted data). Some cryptographic algorithms (e.g. AES) successively modify the cryptographic key during the steps of the cryptographic operation; therefore the decrypt form of the key is different from the encrypt form of the key.
Descriptor	A descriptor is a sequence of commands that causes CAAM to perform cryptographic functions. There are three types of descriptors: job descriptors, shared descriptors, and trusted descriptors. Shared descriptors and trusted descriptors are actually special forms of job descriptors.
Export	Exporting is the act of creating (encapsulating) a Blob. It involves protecting the Blob's privacy, integrity, and optionally, providing protection against replay, utilizing the security mechanisms available to CAAM and the processor. (See <i>Import</i> )
Fail mode	CAAM clears its CSP registers (e.g. key registers) upon entrance to <i>Fail Mode</i> . CAAM enters Fail Mode when the SNVS 's security state machine enters its Fail State. This could be due to the detection of tampering, scan or JTAG testing or due to the failure of a security module.

Table continues on the next page...

**Table A-2. Glossary of terms (continued)**

Term	Description
General memory blob key encryption key	The General Memory Blob Key Encryption Key (GM BKEK) is a 256-bit key used when encrypting cryptographic Blobs exported from general memory (as opposed to CAAM Secure Memory). It is intended for use in protecting the confidentiality and integrity of data exported from general memory. The GM BKEK is derived from the Master Key and a constant embedded in the CAAM Descriptor that initiated the General Memory Blob operation. The GM BKEK is derived so that its value will be different than SM BKEK values. (See <i>SM Blob Key Encryption Key</i> , <i>Master Key</i> .)
Hash	A hash is the message digest resulting from a hashing operation, such as SHA-1, SHA-256 or MD5. A cryptographic hashing operation is a collision-resistant one-way function that yields a fixed-length bit string from a variable length input. A function is collision-resistant if it is difficult to find two input strings that yield the same Message Digest. A function is one-way if it is computationally infeasible to calculate the input, given only the Message Digest.
Import	Importing is the act of retrieving the data from a blob (decapsulating). It involves decrypting the blob and checking its integrity and, optionally, protecting against replay. (See <i>Export</i> )
Job descriptor	The term 'job descriptor' means a descriptor that is not a shared descriptor or a trusted descriptor. Unlike a shared descriptor, a job descriptor can reference another descriptor, and unlike a trusted descriptor, a job descriptor is not signed.
Job descriptor key encryption key	The Job Descriptor Key Encryption Key (JDKEK) is a 256-bit key used to protect the confidentiality of Data Encryption Keys (DEK) referenced by job descriptors. A new JDKEK value is generated by the CAAM's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during job descriptor processing. (See <i>Trusted Descriptor Key Encryption Key</i> .)
Key encryption key	A Key Encryption Key (KEK) is a cryptographic key used to encrypt other cryptographic keys. CAAM supports various KEKs that are used in different circumstances. (See <i>JDKEK</i> , <i>TDKEK</i> )
Link table	A link table is also referred to by the term 'Scatter/Gather Table';
Manager processor	The Manager Processor is the processor that is entrusted with configuring various options in CAAM. This processor is authenticated via its IP bus MID.
Master identifier	The master identifier (MID) is a bus master identifier that is transmitted along with the bus address and data. Since the master identity is used by CAAM to enforce access control, master identity values must be assigned to bus masters in a manner that unambiguously identifies the set of access control permissions that are to be enforced on each bus transaction.
Master key	The master key is a 256-bit secret value that CAAM receives from the SNVS. (See <i>Non-volatile Test Key</i> , <i>OTP Master Key</i> , <i>Zeroizable Secret Key</i> , <i>Secure Memory Blob Key Encryption Key</i> , and <i>General Memory Blob Key Encryption Key</i> .)
Message digest	A message digest (also called a hash) is a fixed-size string that is the result of computing a cryptographic one-way function of some input data.
Non-volatile test key	The Non-volatile Test Key (NVTK) is a 256-bit key hardwired into CAAM. When CAAM is in the Non-Secure Mode CAAM will use the NVTK to derive Blob key encryption keys, rather than using the secret Master Key. The NVTK value is public knowledge, and is the same in every SOC. It is used for known-answer tests when testing the CAAM cryptographic hardware.
Non-secure mode	CAAM's Non-secure Mode is intended to allow CAAM to be tested without compromising the security of sensitive data. In this mode a known version of the BKEK (based on the Non-volatile Test Key) is used for exporting and importing Blobs. Therefore any Blobs exported while in Secure Mode or Trusted Mode cannot be successfully imported while in Non-secure Mode.
OTP master key	The OTP Master Key (OTPMK) is a 256-bit secret value stored in one-time-programmable storage on the SOC. The value is generally written to the one-time-programmable storage while the SOC is in the factory. The OTPMK bits are protected with a lock that, when set, prevents modifying the value. In some configurations SNVS will use the OTPMK to derive the value of the master key that the SNVS supplies over a private bus to CAAM. Its value cannot otherwise be read, sensed or scanned.
Own	The processor that owns a partition can change the access control permissions (and other data) within the partition's SMAPJR and SMAG2/1JR registers. The processor owns all the pages that are currently allocated to partitions owned by the processor. (See <i>Claim</i> )

Table continues on the next page...

**Table A-2. Glossary of terms (continued)**

Term	Description
Page	As used in this Block Guide, the term 'page' refers to a fixed-size portion of Secure Memory. The size of a page is determined at synthesis via a parameter. The size of a Secure Memory page may or may not correspond to the size of virtual memory page as implemented by the processor's MMU. (See <i>Partition</i> )
Partition	A partition of Secure Memory may be unallocated, or it may be allocated to a processor. An AXI bus master can access an owned partition only if the partition's Access Control Permissions are set to allow the access. (See <i>Ownership</i> )
Processor	A processor is a bus master capable of executing software. Processors are given the privilege of claiming Secure Memory partitions, allocating and deallocating pages of Secure Memory, assigning access control permissions, and deallocating partitions.
Public security parameter	A public security parameter (PSP) is security-related public information whose modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)] The Trusted Root Key is a PSP.
Read safe	A read-safe transaction reads a full aligned burst of data, even if not all of the data is needed.
Red blob	A blob whose data input when exporting is assumed to be not encrypted, and whose data output when importing is not encrypted. (See <i>Black Blob</i> )
Red key	A key that is not encrypted. (See <i>Black Key</i> )
Replay	Replay is a type of security attack in which old data is presented by a hacker as if it were new data. For instance, a hacker could replace a new Blob that shows that a software license has expired with an old Blob that indicates that the license is still valid. The term "replay" is sometimes also used to refer to a denial of service attack based upon flooding the system with the same message over and over. If this message is encrypted or cryptographically authenticated, then the attacker may not be able to generate new messages and instead would "replay" a legitimate message that the attacker had snooped from the network.
Secure memory	Secure Memory is an optional component of CAAM that provides access-controlled storage for sensitive data. Secure Memory is divided into multiple partitions, each of which may use different access control settings. (See <i>Claim, Own, Partition, Page</i> )
Secure mode	Secure Mode is the normal operating mode of CAAM. The Security State Machine within the SNVS determines when CAAM is operating in Secure Mode.
Secure nonvolatile storage	SNVS is the Secure Nonvolatile Storage companion logic block to CAAM. It implements a security alarm detection, and maintains a security state machine. SNVS also includes a low-power portion that is intended to be powered from a coin cell battery when main SOC power is off. Whether the main SOC power is on or off, the low-power portion provides non-volatile storage for a Zeroizable Master Key, and accurately maintains a secure real-time clock.
Sensitive data	Sensitive data is data that should be protected against unauthorized disclosure.
Sensitive security parameter	The term 'sensitive security parameters' (SSP) encompasses critical security parameters and public security parameters. [from FIPS PUB 140-3 (DRAFT)]
Secure memory blob key encryption key	The Secure Memory Blob Key Encryption Key (SM BKEK) is a 256-bit key used when encrypting cryptographic Blobs exported from a Secure Memory partition. It is intended for use in protecting the confidentiality and integrity of data exported from CAAM Secure Memory. The SM BKEK is derived from the Master Key and the KMOD field and access control fields from the partition's SMAPJR register and SMAG2/1JR registers. The SM BKEK is derived so that its value will be different than GM BKEK values. (See <i>General Memory Blob Key Encryption Key, Master Key</i> )
SEQ	Sequence. For most memory referencing descriptor commands CAAM implements an auto-incrementing addressing mode using sequence input address and sequence output address registers. This is intended to facilitate the processing of cryptographic networking protocols.
Shared descriptor	A shared descriptor is a special type of job descriptor that can be executed only when it is referenced by another descriptor. Shared descriptors are intended to contain data, such as keys and sequence numbers, that are shared by two or more other descriptors.

*Table continues on the next page...*

**Table A-2. Glossary of terms (continued)**

Term	Description
Trusted descriptor	A trusted descriptor is a special type of job descriptor that has some additional access privileges and some additional security protections. When CAAM is executing a job descriptor, CAAM can use the data within Secure Memory key partitions (partitions that have SMAPJR[K_D] set) only as keys in cryptographic operations. When CAAM is executing a trusted descriptor, CAAM can use the data within key partitions as data in cryptographic operations. This feature allows trusted descriptors to be used in key derivation and key generation operations. Trusted descriptors are protected from modification by means of a signature over the descriptor. CAAM verifies the signature before executing the trusted descriptor, and aborts execution if the signature is incorrect. (See <i>Trusted Descriptor Signing Key</i> )
Trusted descriptor signing key	The <i>Trusted Descriptor Signing Key</i> (TDSK) is a 256-bit key used in HMAC-SHA-256 to sign and verify the signature over trusted descriptors. A new TDSK value is generated by the CAAM's RNG at each POR, and is used throughout the current power-on cycle. CAAM will allow TDSK to be used to sign a new trusted descriptor only if the descriptor is submitted via a Job Ring that has AMTD set in its JRMID register. Otherwise, CAAM will use TDSK only to verify the signature over a trusted descriptor, or to update the signature on an existing trusted descriptor that has modified itself during its execution.
Trusted descriptor key encryption key	The Trusted Descriptor Key Encryption Key (TDKEK) is a 256-bit key that can be used to protect the confidentiality of Data Encryption Keys (DEKs) referenced by trusted descriptors. A new TDKEK value is generated by the CAAM's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during trusted descriptor processing. (See <i>Job Descriptor Key Encryption Key</i> )
Trusted mode	Trusted Mode is a special operating mode of CAAM. The Security State Machine within the SNVS determines when CAAM is operating in Trusted Mode. This mode is implemented so that trusted boot-time software, or a hypervisor or TrustZone Secure World software can store data in and retrieve data from Trusted Mode Blobs that are not accessible to software running while CAAM is in Secure Mode or NonSecure Mode.
Trusted root key	The Trusted Root Key is a public signature key used by HAB to verify the signature over the Command Sequence File. The key could be RSA (probably 2048 bits) or ECC-DSA (probably 511 bits). The integrity and authenticity of this key is protected by placing a SHA-256 hash of this key in fuses on the SOC. The fuses are located in a bank with a lock fuse that, when set, prevents any changes to the hash value.
Unallocated	An unallocated partition is not associated with any processor. It is available for allocation. A processor that has access to the Secure Memory can write into the partition's SMAPJR register to claim the partition. Only the processor that owns the partition can de-allocate that partition. The term ' <i>unallocated</i> ' is used interchangeably with the term ' <i>unclaimed</i> '.
Unclaimed	The term ' <i>unclaimed</i> ' is used interchangeably with the term ' <i>unallocated</i> '.
Word	A word of memory or a one-word register contains 32 bits.
Write safe	A write-safe transaction writes 0s to addresses past the targeted locations up to the next 8, 16, 32 or 64-byte address boundary, depending upon the offset within the cacheline.
Zeroizable secret key	The Zeroizable Secret Key (ZSK) is a 256-bit key stored in a register in the low-power domain of SNVS. In some configurations and security states SNVS will use the ZSK to derive the value of the Master Key that SNVS supplies over the snvs_master_key signal to CAAM. Its value cannot otherwise be read or scanned. The value can be generated by the CAAM RNG, and can be loaded automatically by hardware. The value can be zeroized when a tamper event is detected. (See <i>Master Key</i> and <i>Master Key</i> .)
Zeroize	A set of data storage locations is <i>zeroized</i> by overwriting the storage locations with a value (not necessarily 0) that is independent of the previous content of the storage locations.

## Appendix B

# CAAM Register Descriptions

### B.1 CAAM memory map/register definition

The CAAM memory map is broken into the following 4 Kbyte blocks, listed in this table.

**Table B-1. Memory map structure**

Block number	Included registers
0	General registers (for example, configuration, control, debugging, and RNG)
1–2	Job Ring registers (JR0–1) and Secure Memory registers
8	Descriptor controller (DECO) and CHA control block (CCB) registers

All reads of undefined and write-only addresses always return zero. Writes to undefined and read-only addresses are ignored. CAAM will never generate a transfer error on the register bus. Although many of the CAAM registers hold more than 32 bits, the register addresses shown in the Memory Map below and the diagrams in [Section B.1.2, “Register Summary,”](#) represent how these registers are accessed over the register bus as 32-bit words.

#### NOTE

64 KB of address space is reserved for accessing CAAM registers. This address space is divided into 16 4 KB pages to match the access granularity of most MMUs. Registers that are intended to be accessed by a specific processor or process are grouped into one of these 16 pages, so that access to these registers can be restricted via CAAM’s own MID-based access control mechanism, via the CPU’s MMU or via a system MMU. For instance, the general configuration and status registers are located within page 0, and are intended to be accessed only by a specially privileged process. The registers that control each Job Ring are located in a separate page so that access to each Job Ring can be restricted to a particular process. Some registers, such as the version ID registers, are intended to be shared among processes. Rather than require each CAAM driver process to have two MMU page entries, one page for its private registers and one for the shared registers, CAAM “aliases” these shared registers into the upper section of each of the 16 pages. Reading

any one of the address aliases for the same register returns the same information. Some of these aliased registers are writable, so access to these registers may require that software implement a concurrency control construct, as would be the case with any register that is read/write accessible by multiple processes. In versions of CAAM that incorporate Secure Memory, a few read/write registers (i.e. the SMAPJR and SMAGJR registers) are aliased to each of the Job Ring register pages. This is done so that Secure Memory partitions can be claimed on a first-come first-served basis. The first of the Job Ring owners to write to a partition's SMAPJR register claims that partition, and the SMAPJR and SMAGJR registers for that partition become inaccessible to other Job Ring owners.

### B.1.1 Memory Map

CAAM's Secure Memory is accessible via the AXI bus at the addresses indicated in the following table.

**Table B-2. CAAM Memory Map: Secure Memory AXI Slave Interface**

Offset or Address	Register	Access
<b>Secure Memory</b>		
MEMBASE_ + N*PartitionSize(in Bytes) to MEMBASE_ + N*PartitionSize + PartitionSize-1	Memory Partition N.	R/W

Most of CAAM's configuration registers are accessible in block 0 of CAAM's register space, as indicated in the following table. These registers are intended to be accessed by specially privileged software (e.g. boot software, hypervisor, secure operating system).

**Table B-3. CAAM Memory Map: Configuration and Special Key Registers**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol “&amp;” following a register’s offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with ‘#’ is aliased to multiple addresses, each located in a different 4k-byte page.</p>			
<p>Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details.</p>			
0xBASE_0004	MCFG — Master Configuration Register	R/W	<a href="#">B.2.1.1/B-461</a>
0xBASE_000C	SCFGR — Security Configuration Register	R/W	<a href="#">B.2.1.2/B-463</a>
0xBASE_009C	DECORR — DECO Request Register	R/W	<a href="#">B.2.1.4/B-468</a>
0xBASE_00A0	DECO0MIDR — DECO 0 MID Register	R/W	<a href="#">B.2.1.5/B-469</a>
0xBASE_0120	DAR — DECO Availability Register	R/W	<a href="#">B.2.1.6/B-470</a>
0xBASE_0400 - 0xBASE_041F	JDKEKR — Job Descriptor Key Encryption Key Register, bits 255-224, 223-192, 191-160, 159-128, 127-96, 95-64, 63-32, 31-0, in order of increasing addresses	Non-Secure Mode R/W	<a href="#">B.2.1.8/B-472</a>
0xBASE_0420 - 0xBASE_043F	TDKEKR — Trusted Descriptor Key Encryption Key Register, bits 255-224, 223-192, 191-160, 159-128, 127-96, 95-64, 63-32, 31-0, in order of increasing addresses	Non-Secure Mode R/W	<a href="#">B.2.1.9/B-473</a>
0xBASE_0440 - 0xBASE_045F	TDSKR — Trusted Descriptor Signing Key Register, bits 255-224, 223-192, 191-160, 159-128, 127-96, 95-64, 63-32, 31-0, in order of increasing addresses	Non-Secure Mode R/W	<a href="#">B.2.1.10/B-474</a>
0xBASE_04E0&	SKNR — Secure Key Nonce Register	Non-Secure Mode R/W	<a href="#">B.2.1.11/B-475</a>

The registers used for testing CAAM’s Random Number Generator are listed in the following table. These registers do not need to be accessed for normal use of the RNG.

**Table B-4. CAAM Memory Map: RNG Registers**

Offset or Address	Register	Access	Section/Page
<b>The following registers are used to test the RNG. Since the RNG is accessible via Job Descriptor commands, it is not necessary to access these registers during normal operations.</b>			
Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details. <b>Any register whose offset is marked with a '@x' subscript is accessible at the specified address when field PRGM=x in the RTMCTL register (0xBASE_0600).</b>			
0xBASE_0600	RTMCTL — RNG TRNG Miscellaneous Control Register	R/W	<a href="#">B.2.2.1/B-476</a>
0xBASE_0604	RTSCMISC — RNG TRNG Statistical Check Miscellaneous Register	R/W	<a href="#">B.2.2.2/B-478</a>
0xBASE_0608	RTPKRRNG — RNG TRNG Poker Range Register	R/W	<a href="#">B.2.2.3/B-479</a>
0xBASE_060C@1	RTPKRMAX — RNG TRNG Poker Maximum Limit Register	R/W	<a href="#">B.2.2.3/B-479</a>
0xBASE_060C@0	RTPKRSQ — RNG TRNG Poker Square Calculation Result Register	R/W	<a href="#">B.2.2.5/B-480</a>
0xBASE_0610	RTSDCTL — RNG TRNG Seed Control Register	R/W	<a href="#">B.2.2.6/B-481</a>
0xBASE_0614@1	RTSBLIM — RNG TRNG Sparse Bit Limit Register	R/W	<a href="#">B.2.2.7/B-482</a>
0xBASE_0614@0	RTTOTSAM — RNG TRNG Total Samples Register	R/W	<a href="#">B.2.2.8/B-483</a>
0xBASE_0618	RTFRQMIN — RNG TRNG Frequency Count Minimum Limit Register	R/W	<a href="#">B.2.2.9/B-483</a>
0xBASE_061C@1	RTFRQMAX — RNG TRNG Frequency Count Maximum Limit Register	R/W	<a href="#">B.2.2.10/B-484</a>
0xBASE_061C@0	RTFRQCNT — RNG TRNG Frequency Count Register	R/W	<a href="#">B.2.2.11/B-485</a>
0xBASE_0620@1	RTSCML — RNG TRNG Statistical Check Monobit Limit Register	R/W	<a href="#">B.2.2.12/B-486</a>
0xBASE_0620@0	RTSCMC — RNG TRNG Statistical Check Monobit Count Register	R/W	<a href="#">B.2.2.13/B-487</a>
0xBASE_0624@1	RTSCR1L — RNG TRNG Statistical Check Run Length 1 Limit Register	R/W	<a href="#">B.2.2.14/B-487</a>
0xBASE_0624@0	RTSCR1C — RNG TRNG Statistical Check Run Length 1 Count Register	R/W	<a href="#">B.2.2.15/B-488</a>
0xBASE_0628@1	RTSCR2L — RNG TRNG Statistical Check Run Length 2 Limit Register	R/W	<a href="#">B.2.2.16/B-489</a>
0xBASE_0628@0	RTSCR2C — RNG TRNG Statistical Check Run Length 2 Count Register	R/W	<a href="#">B.2.2.17/B-490</a>
0xBASE_062C@1	RTSCR3L — RNG TRNG Statistical Check Run Length 3 Limit Register	R/W	<a href="#">B.2.2.18/B-491</a>
0xBASE_062C@0	RTSCR3C — RNG TRNG Statistical Check Run Length 3 Count Register	R/W	<a href="#">B.2.2.19/B-492</a>
0xBASE_0630@1	RTSCR4L — RNG TRNG Statistical Check Run Length 4 Limit Register	R/W	<a href="#">B.2.2.20/B-493</a>
0xBASE_0630@0	RTSCR4C — RNG TRNG Statistical Check Run Length 4 Count Register	R/W	<a href="#">B.2.2.21/B-494</a>
0xBASE_0634@1	RTSCR5L — RNG TRNG Statistical Check Run Length 5 Limit Register	R/W	<a href="#">B.2.2.22/B-495</a>
0xBASE_0634@0	RTSCR5C — RNG TRNG Statistical Check Run Length 5 Count Register	R/W	<a href="#">B.2.2.23/B-496</a>
0xBASE_0638@1	RTSCR6PL — RNG TRNG Statistical Check Run Length 6+ Limit Register	R/W	<a href="#">B.2.2.24/B-497</a>
0xBASE_0638@0	RTSCR6PC — RNG TRNG Statistical Check Run Length 6+ Count Register	R/W	<a href="#">B.2.2.25/B-498</a>
0xBASE_063C@0	RTSTATUS — RNG TRNG Status Register	R	<a href="#">B.2.2.20/B-493</a>
0xBASE_0640	RTENT0 — RNG TRNG Entropy Read Register, bits [383:352]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0644	RTENT1 — RNG TRNG Entropy Read Register, bits [351:320]	R	<a href="#">B.2.2.27/B-501</a>

**Table B-4. CAAM Memory Map: RNG Registers**

Offset or Address	Register	Access	Section/Page
0xBASE_0648	RTENT2 — RNG TRNG Entropy Read Register, bits [319:288]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_064C	RTENT3 — RNG TRNG Entropy Read Register, bits [287:256]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0650	RTENT4 — RNG TRNG Entropy Read Register, bits [255:224]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0654	RTENT5 — RNG TRNG Entropy Read Register, bits [223:192]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0658	RTENT6 — RNG TRNG Entropy Read Register, bits [191:160]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_065C	RTENT7 — RNG TRNG Entropy Read Register, bits [159:128]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0660	RTENT8 — RNG TRNG Entropy Read Register, bits [127:96]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0664	RTENT9 — RNG TRNG Entropy Read Register, bits [95:64]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0668	RTENT10 — RNG TRNG Entropy Read Register, bits [63:32]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_066C	RTENT11 — RNG TRNG Entropy Read Register, bits [31:0]	R	<a href="#">B.2.2.27/B-501</a>
0xBASE_0680	RTPKRCNT10 — RNG TRNG Statistical Check Poker Count 1 and 0 Register	R	<a href="#">B.2.2.28/B-502</a>
0xBASE_0684	RTPKRCNT32 — RNG TRNG Statistical Check Poker Count 3 and 2 Register	R	<a href="#">B.2.2.29/B-503</a>
0xBASE_0688	RTPKRCNT54 — RNG TRNG Statistical Check Poker Count 5 and 4 Register	R	<a href="#">B.2.2.30/B-504</a>
0xBASE_068C	RTPKRCNT76 — RNG TRNG Statistical Check Poker Count 7 and 6 Register	R	<a href="#">B.2.2.31/B-504</a>
0xBASE_0690	RTPKRCNT98 — RNG TRNG Statistical Check Poker Count 9 and 8 Register	R	<a href="#">B.2.2.32/B-505</a>
0xBASE_0694	RTPKRCNTBA — RNG TRNG Statistical Check Poker Count B and A Register	R	<a href="#">B.2.2.33/B-506</a>
0xBASE_0698	RTPKRCNTDC — RNG TRNG Statistical Check Poker Count D and C Register	R	<a href="#">B.2.2.34/B-507</a>
0xBASE_069C	RTPKRCNTFE — RNG TRNG Statistical Check Poker Count F and E Register	R	<a href="#">B.2.2.35/B-508</a>
0xBASE_06C0	RDSTA — RNG DRNG Status Register	R	<a href="#">B.2.2.36/B-509</a>
0xBASE_06D0	RDINT0 — RNG DRNG State Handle 0 Reseed Interval Register	R	<a href="#">B.2.2.37/B-511</a>
0xBASE_06D4	RDINT1 — RNG DRNG State Handle 1 Reseed Interval Register	R	<a href="#">B.2.2.38/B-511</a>
0xBASE_06E0	RDHCNTL — RNG DRNG Hash Control Register	R/W	<a href="#">B.2.2.39/B-512</a>
0xBASE_06E4	RDHDIG — RNG DRNG Hash Digest Register	R	<a href="#">B.2.2.40/B-513</a>
0xBASE_06E8	RDHBUF — RNG DRNG Hash Buffer Register	W	<a href="#">B.2.2.41/B-514</a>
0xBASE_0BF8 &#	CAAMVID — CAAM Version ID Register	R	<a href="#">B.2.5.12/B-554</a>

The following table lists registers that are used to debug the execution of Job Descriptors launched from a Job Ring. These registers would not be accessed during normal CAAM operation.

**Table B-5. CAAM Memory Map: Holding Tank and Job Ring Debugging Registers**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol "&amp;" following a register's offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with '#' is aliased to multiple addresses.</p>			
<p>Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details.</p>			
0xBASE_0C00 &	HT0_JOB_DESC — Holding Tank 0 Job Descriptor	R	<a href="#">B.2.3.1/B-514</a>
0xBASE_0C08 &	HT0_SHR_DESC — Holding Tank 0 Shared Descriptor	R	<a href="#">B.2.3.2/B-516</a>
0xBASE_0C10 &	HT0_JQ_CTRL — Holding Tank 0 Job Queue Control	R	<a href="#">B.2.3.3/B-517</a>
0xBASE_0C18 &	HT0_STATUS — Holding Tank 0 Status	R	<a href="#">B.2.3.4/B-519</a>
0xBASE_0D70 &	JRAAA0 — Job Ring Address—Array Address 0 Register	R	<a href="#">B.2.3.5/B-520</a>
0xBASE_0D78 &	JRAAA1 — Job Ring Address—Array Address 1 Register	R	<a href="#">B.2.3.5/B-520</a>
0xBASE_0D80 &	JRAAA2 — Job Ring Address—Array Address 2 Register	R	<a href="#">B.2.3.5/B-520</a>
0xBASE_0D88 &	JRAAA3 — Job Ring Address—Array Address 3 Register	R	<a href="#">B.2.3.5/B-520</a>
0xBASE_0DB4	JRAAVS — Job Ring Address—Array Valid/Source Register	R	<a href="#">B.2.3.6/B-522</a>
0xBASE_0DB8 &	JRJIDU — Job Ring Job IDs in Use Register	R	<a href="#">B.2.3.7/B-523</a>
0xBASE_0DC0	JRJDJIF0 — Job Ring Job-Done Job ID FIFO 0	R	<a href="#">B.2.3.8/B-526</a>
0xBASE_0DC4	JRJDJIF1 — Job Ring Job-Done Job ID FIFO 1	R	<a href="#">B.2.3.8/B-526</a>
0xBASE_0DC8	JRJDJIF2 — Job Ring Job-Done Job ID FIFO 2	R	<a href="#">B.2.3.8/B-526</a>
0xBASE_0DCC	JRJDJIF3 — Job Ring Job-Done Job ID FIFO 3	R	<a href="#">B.2.3.8/B-526</a>
0xBASE_0DE0	JRJDSF0 — Job Ring Job-Done Source FIFO 0	R	<a href="#">B.2.3.9/B-529</a>
0xBASE_0DE4	JRJDSF1 — Job Ring Job-Done Source FIFO 1	R	<a href="#">B.2.3.9/B-529</a>
0xBASE_0DF0	JRJDVF — Job Ring Job-Done Valid FIFO	R	—
0xBASE_0DF4	JRJDS — Job Ring Job-Done Status Register	R	<a href="#">B.2.3.11/B-533</a>

The following table lists registers that are used to obtain information concerning AXI bus faults, and registers used to determine the version of CAAM and the versions of its internal components.

**Table B-6. CAAM Memory Map: Fault and Version ID Registers**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol “&amp;” following a register’s offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with ‘#’ is aliased to multiple addresses.</p>			
<p>Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details.</p>			
0xBASE_0FA0 &#	CRNR — CHA Revision Number Register	R	<a href="#">B.2.5.1/B-535</a>
0xBASE_0FA8 &#	CTPR — Compile Time Parameters Register	R	<a href="#">B.2.5.2/B-537</a>
0xBASE_0FB0 #	SMSTA — Secure Memory Status Register	R	<a href="#">B.2.5.3/B-540</a>
0xBASE_0FB8 #	SMPO — Secure Memory Partition Owners Register	R	<a href="#">B.2.5.4/B-541</a>
0xBASE_0FC4 #	FAR — Fault Address Register	R	<a href="#">B.2.5.5/B-543</a>
0xBASE_0FCC #	FADR — Fault Address Detail Register	R	<a href="#">B.2.5.6/B-544</a>
0xBASE_0FD4 #	CSTA — CAAM Status Register	R	<a href="#">B.2.5.7/B-546</a>
0xBASE_0FD8 &#	SMVID — Secure Memory Version ID Register	R	<a href="#">B.2.5.8/B-548</a>
0xBASE_0FE4 #	CCBVID — CHA Cluster Block Version ID Register	R	<a href="#">B.2.5.9/B-549</a>
0xBASE_0FE8 &#	CHAVID — CHA Version ID Register	R	<a href="#">B.2.5.10/B-550</a>
0xBASE_0FF0 &#	CHANUM — CHA Number Register	R	<a href="#">B.2.5.11/B-552</a>
0xBASE_0FF8 &#	CAAMVID — CAAM Version ID Register	R	<a href="#">B.2.5.12/B-554</a>

The following tables list the registers associated with the Job Rings, including the Job Ring Secure Memory registers, and registers used to obtain information concerning AXI bus faults and the version of CAAM and its internal components.

**Table B-7. CAAM Memory Map: Job Ring 0, Secure Memory, Fault and Version ID Registers**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol "&amp;" following a register's offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with '#' is aliased to multiple addresses.</p>			
<p>Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details.</p>			
0xBASE_1004	IRBAR0 — Input Ring Base Address Register for Job Ring 0	R/W	<a href="#">B.2.6.1/B-556</a>
0xBASE_100C	IRSR0 — Input Ring Size Register for Job Ring 0	R/W	<a href="#">B.2.6.2/B-557</a>
0xBASE_1014	IRSAR0 — Input Ring Slots Available Register for Job Ring 0	R	<a href="#">B.2.6.3/B-558</a>
0xBASE_101C	IRJAR0 — Input Ring Jobs Added Register for Job Ring 0	R/W	<a href="#">B.2.6.4/B-559</a>
0xBASE_1024	ORBAR0 — Output Ring Base Address Register for Job Ring 0	R/W	<a href="#">B.2.6.5/B-560</a>
0xBASE_102C	ORSR0 — Output Ring Size Register for Job Ring 0	R/W	<a href="#">B.2.6.6/B-561</a>
0xBASE_1034	ORJRR0 — Output Ring Jobs Removed Register for Job Ring 0	R/W	<a href="#">B.2.6.7/B-562</a>
0xBASE_103C	ORSFR0 — Output Ring Slots Full Register for Job Ring 0	R	<a href="#">B.2.6.8/B-563</a>
0xBASE_1044	JROSR0 — Job Ring Output Status Register for Job Ring 0	R	<a href="#">B.2.6.9/B-564</a>
0xBASE_104C	JRINTR0 — Job Ring Interrupt Status Register for Job Ring 0	R/W	<a href="#">B.2.6.10/B-572</a>
0xBASE_1050&	JRCFGR0 — Job Ring Configuration Register for Job Ring 0	R/W	<a href="#">B.2.6.11/B-573</a>
0xBASE_105C	IRRIR0 — Input Ring Read Index Register for Job Ring 0	R	<a href="#">B.2.6.12/B-579</a>
0xBASE_1064	ORWIR0 — Output Ring Write Index Register for Job Ring 0	R	<a href="#">B.2.6.13/B-580</a>
0xBASE_106C	JRCR0 — Job Ring Command Register for Job Ring 0	R/W	<a href="#">B.2.6.14/B-581</a>
0xBASE_10F4	SMCJR0 — Secure Memory Command Register for Job Ring 0	W	<a href="#">B.2.7.1/B-582</a>
0xBASE_10FC	SMCSJR0 — Secure Memory Command Status Register for Job Ring 1	R	<a href="#">B.2.7.2/B-584</a>
0xBASE_11x4 #	SMAPJR0x — Secure Memory Access Permissions Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.3/B-587</a>
0xBASE_11x8 #	SMAG2JR0x — Secure Memory Access Group 2 Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.4/B-593</a>
0xBASE_11xC #	SMAG1JR0x — Secure Memory Access Group 1 Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.4/B-593</a>
0xBASE_1FA0 &#	CRNR — CHA Revision Number Register	R	<a href="#">B.2.5.1/B-535</a>
0xBASE_1FA8 &#	CTPR — Compile Time Parameters Register	R	<a href="#">B.2.5.2/B-537</a>
0xBASE_1FB0 #	SMSTA — Secure Memory Status Register	R	<a href="#">B.2.5.3/B-540</a>
0xBASE_1FB8 #	SMPO — Secure Memory Partition Owners Register	R	<a href="#">B.2.5.4/B-541</a>
0xBASE_1FC4 #	FAR — Fault Address Register	R	<a href="#">B.2.5.5/B-543</a>

**Table B-7. CAAM Memory Map: Job Ring 0, Secure Memory, Fault and Version ID Registers**

Offset or Address	Register	Access	Section/Page
0xBASE_1FC8 #			
0xBASE_1FCC #	FADR — Fault Address Detail Register	R	<a href="#">B.2.5.6/B-544</a>
0xBASE_1FD4 #	CSTA — CAAM Status Register	R	<a href="#">B.2.5.7/B-546</a>
0xBASE_1FD8 &#	SMVID — Secure Memory Version ID Register	R	<a href="#">B.2.5.8/B-548</a>
0xBASE_1FE4 #	CCBVID — CHA Cluster Block Version ID Register	R	<a href="#">B.2.5.9/B-549</a>
0xBASE_1FE8 &#	CHAVID — CHA Version ID Register	R	<a href="#">B.2.5.10/B-550</a>
0xBASE_1FF0 &#	CHANUM — CHA Number Register	R	<a href="#">B.2.5.11/B-552</a>
0xBASE_1FF8 &#	CAAMVID — CAAM Version ID Register	R	<a href="#">B.2.5.12/B-554</a>

**Table B-8. CAAM Memory Map: Job Ring 1, Secure Memory, Fault and Version ID Registers**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol “&amp;” following a register’s offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with ‘#’ is aliased to multiple addresses.</p>			
<p>Note that access to most registers is limited to specific Bus Master Identities. See the register descriptions for details.</p>			
0xBASE_2004	IRBAR1 — Input Ring Base Address Register for Job Ring 1	R/W	<a href="#">B.2.6.1/B-556</a>
0xBASE_200C	IRSR1 — Input Ring Size Register for Job Ring 1	R/W	<a href="#">B.2.6.2/B-557</a>
0xBASE_2014	IRSAR1 — Input Ring Slots Available Register for Job Ring 1	R	<a href="#">B.2.6.3/B-558</a>
0xBASE_201C	IRJAR1 — Input Ring Jobs Added Register for Job Ring 1	R/W	<a href="#">B.2.6.4/B-559</a>
0xBASE_2024	ORBAR1 — Output Ring Base Address Register for Job Ring 1	R/W	<a href="#">B.2.6.5/B-560</a>
0xBASE_202C	ORSR1 — Output Ring Size Register for Job Ring 1	R/W	<a href="#">B.2.6.6/B-561</a>
0xBASE_2034	ORJRR1 — Output Ring Jobs Removed Register for Job Ring 1	R/W	<a href="#">B.2.6.7/B-562</a>
0xBASE_203C	ORSFR1 — Output Ring Slots Full Register for Job Ring 1	R	<a href="#">B.2.6.8/B-563</a>
0xBASE_2044	JROSR1 — Job Ring Output Status Register for Job Ring 1	R	<a href="#">B.2.6.9/B-564</a>
0xBASE_204C	JRINTR1 — Job Ring Interrupt Status Register for Job Ring 1	R/W	<a href="#">B.2.6.10/B-572</a>
0xBASE_2050&	JRCFGR1 — Job Ring Configuration Register for Job Ring 1	R/W	<a href="#">B.2.6.11/B-573</a>
0xBASE_205C	IRRIR1 — Input Ring Read Index Register for Job Ring 1	R	<a href="#">B.2.6.12/B-579</a>

**Table B-8. CAAM Memory Map: Job Ring 1, Secure Memory, Fault and Version ID Registers**

Offset or Address	Register	Access	Section/Page
0xBASE_2064	ORWIR1 — Output Ring Write Index Register for Job Ring 1	R	<a href="#">B.2.6.13/B-580</a>
0xBASE_206C	JRCR1 — Job Ring Command Register for Job Ring 1	R/W	<a href="#">B.2.6.14/B-581</a>
0xBASE_20F4	SMCJR1 — Secure Memory Command Register for Job Ring 1	W	<a href="#">B.2.7.1/B-582</a>
0xBASE_20FC	SMCSJR1 — Secure Memory Command Status Register for Job Ring 1	R	<a href="#">B.2.7.2/B-584</a>
0xBASE_21x4 #	SMAPJR1x — Secure Memory Access Permissions Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.3/B-587</a>
0xBASE_21x8 #	SMAG2JR1x — Secure Memory Access Group 2 Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.4/B-593</a>
0xBASE_21xC #	SMAG1JR1x — Secure Memory Access Group 1 Register for Job Ring 1 & Partition x	R/W	<a href="#">B.2.7.4/B-593</a>
0xBASE_2FA0 &#	CRNR — CHA Revision Number Register	R	<a href="#">B.2.5.1/B-535</a>
0xBASE_2FA8 &#	CTPR — Compile Time Parameters Register	R	<a href="#">B.2.5.2/B-537</a>
0xBASE_2FB0 #	SMSTA — Secure Memory Status Register	R	<a href="#">B.2.5.3/B-540</a>
0xBASE_2FB8 #	SMPO — Secure Memory Partition Owners Register	R	<a href="#">B.2.5.4/B-541</a>
0xBASE_2FC4 #	FAR — Fault Address Register	R	<a href="#">B.2.5.5/B-543</a>
0xBASE_2FC8 #			
0xBASE_2FCC #	FADR — Fault Address Detail Register	R	<a href="#">B.2.5.6/B-544</a>
0xBASE_2FD4 #	CSTA — CAAM Status Register	R	<a href="#">B.2.5.7/B-546</a>
0xBASE_2FD8 &#	SMVID — Secure Memory Version ID Register	R	<a href="#">B.2.5.8/B-548</a>
0xBASE_2FE4 #	CCBVID — CHA Cluster Block Version ID Register	R	<a href="#">B.2.5.9/B-549</a>
0xBASE_2FE8 &#	CHAVID — CHA Version ID Register	R	<a href="#">B.2.5.10/B-550</a>
0xBASE_2FF0 &#	CHANUM — CHA Number Register	R	<a href="#">B.2.5.11/B-552</a>
0xBASE_2FF8 &#	CAAMVID — CAAM Version ID Register	R	<a href="#">B.2.5.12/B-554</a>

The following table lists registers used to initialize CAAM’s Real Time Integrity Checker, and registers used to obtain information concerning AXI bus faults and the version of CAAM and its internal components.

The following table lists registers used to debug the execution of job descriptors by single-stepping them in DECO 0. These registers would not be accessed during normal CAAM operation.

**Table B-9. CAAM Memory Map: DECO 0 and CCB 0**

Offset or Address	Register	Access	Section/Page
<p>The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. The symbol “&amp;” following a register’s offset indicates that the register contains 64 bits, and the most-significant half is accessed at the indicated offset while the least-significant half is accessed at that offset plus 4 bytes. Address ranges are indicated for those registers that are larger than 64 bits. All addresses not shown are reserved. Note that any register whose offset is marked with ‘#’ is aliased to multiple addresses.</p>			
<p>The registers below that are indicated by shading are accessible only when direct DECO access has been granted via the DECORR register, and can be accessed only by the Bus Master Identity specified in the DECO0MID register.</p>			
0xBASE_8004	C0C1MR — CCB 0 Class 1 Mode Register	R/W	<a href="#">B.2.8.1/B-597</a>
0xBASE_800C	C0C1KSR — CCB 0 Class 1 Key Size Register	R/W	<a href="#">B.2.8.2/B-603</a>
0xBASE_8010 &	C0C1DSR — CCB 0 Class 1 Data Size Register	R/W	<a href="#">B.2.8.3/B-604</a>
0xBASE_801C	C0C1ICVSR — CCB 0 Class 1 ICV Size Register	R/W	<a href="#">B.2.8.4/B-605</a>
0xBASE_8034	C0CCTRL — CCB 0 CHA Command Register	W	<a href="#">B.2.8.5/B-606</a>
0xBASE_8038	C0CIRQ — CCB 0 Interrupt Status Register	R/W	<a href="#">B.2.8.6/B-608</a>
0xBASE_8040	C0CWR — CCB 0 Clear Written Register	R/W	<a href="#">B.2.8.7/B-610</a>
0xBASE_8048 &	C0CSTA — CCB 0 Status and Error Register	R	<a href="#">B.2.8.8/B-612</a>
0xBASE_805C	C0AADSZR — CCB 0 AAD Size Register	R/W	<a href="#">B.2.8.9/B-616</a>
0xBASE_8100 - 0xBASE_8147	C0C1CTXR — CCB 0 Class 1 Context Register, bits 575-544, 543-512, 511-480, 479-448, 447-416, 415-384, 383-352, 351-320, 319-288, 287-256, 255-224, 223-192, 191-160, 159-128, 127-96, 95-64, 63-32, 31-0, in order of increasing 4-byte addresses	R/W	<a href="#">B.2.8.10/B-616</a>
0xBASE_8200 - 0xBASE_827F	C0C1KEYR — CCB 0 Class 1 Key Registers, bits 1023-992, 991-960, 959-928, 927-896, 895-864, 863-832, 831-800, 799-768, 767-736, 735-704, 703-672, 671-640, 639-608, 607-576, 575-544, 543-512, 511-480, 479-448, 447-416, 415-384, 383-352, 351-320, 319-288, 287-256, 255-224, 223-192, 191-160, 159-128, 127-96, 95-64, 63-32, 31-0, in order of increasing 4-byte addresses	R/W	<a href="#">B.2.8.11/B-617</a>
0xBASE_8404	C0C2MR — CCB 0 Class 2 Mode Register	R/W	<a href="#">B.2.8.12/B-618</a>
0xBASE_840C	C0C2KSR — CCB 0 Class 2 Key Size Register	R/W	<a href="#">B.2.8.13/B-619</a>
0xBASE_8410 &	C0C2DSR — CCB 0 Class 2 Data Size Register	R/W	<a href="#">B.2.8.14/B-620</a>
0xBASE_841C	C0C2ICVSZR — CCB 0 Class 2 ICV Size Register	R/W	<a href="#">B.2.8.15/B-622</a>
0xBASE_87D0	C0NFIFO — CCB 0 iNformation FIFO	W	<a href="#">B.2.8.18/B-624</a>
0xBASE_87E0	C0IFIFO — CCB 0 Input Data FIFO	W	<a href="#">B.2.8.19/B-627</a>
0xBASE_87F0 &	C0OFIFO — CCB 0 Output Data FIFO; most-significant 32-bits of each 64-bit FIFO entry is read from 0xBASE_87F0h, least-significant 32-bits of each 64-bit FIFO entry is read from 0xBASE_87F4h	R	<a href="#">B.2.8.20/B-628</a>
0xBASE_8800 &	D0JQCR — DECO 0 Job Queue Control Register	R/W	<a href="#">B.2.8.21/B-630</a>
0xBASE_8808	D0DAR — DECO 0 Descriptor Address Register	R/W	<a href="#">B.2.8.22/B-632</a>

**Table B-9. CAAM Memory Map: DECO 0 and CCB 0**

Offset or Address	Register	Access	Section/Page
0xBASE_8810	D0OPSTA — DECO 0 Operation Status Register	R	<a href="#">B.2.8.23/B-632</a>
0xBASE_8820 &			
0xBASE_8840 &	D0MTH0 — DECO 0 Math Register 0	R/W	<a href="#">B.2.8.24/B-635</a>
0xBASE_8848 &	D0MTH1 — DECO 0 Math Register 1	R/W	<a href="#">B.2.8.24/B-635</a>
0xBASE_8850 &	D0MTH2 — DECO 0 Math Register 2	R/W	<a href="#">B.2.8.24/B-635</a>
0xBASE_8858 &	D0MTH3 — DECO 0 Math Register 3	R/W	<a href="#">B.2.8.24/B-635</a>
0xBASE_8880-F	D0GTR0 — DECO 0 Gather Table Register 0	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8890-F	D0GTR1 — DECO 0 Gather Table Register 1	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_88A0-F	D0GTR2 — DECO 0 Gather Table Register 2	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_88B0-F	D0GTR3 — DECO 0 Gather Table Register 3	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8900-F	D0STR0 — DECO 0 Scatter Table Register 0	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8910-F	D0STR1 — DECO 0 Scatter Table Register 1	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8920-F	D0STR2 — DECO 0 Scatter Table Register 2	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8930-F	D0STR3 — DECO 0 Scatter Table Register 3	R/W	<a href="#">B.2.8.25/B-637</a>
0xBASE_8A00 - 0xBASE_8AFF	D0DESB DECO 0 Descriptor Buffer	R/W	<a href="#">B.2.8.35/B-649</a>
0xBASE_8E00	D0DJR — DECO 0 DBG_JOB_REG	R	<a href="#">B.2.8.27/B-641</a>
0xBASE_8E04	D0DDR — DECO 0 DBG_DBG_REG	R	<a href="#">B.2.8.28/B-642</a>
0xBASE_8E08&	D0DJP — DECO 0 DBG_JOB_PTR	R	<a href="#">B.2.8.29/B-644</a>
0xBASE_8E10&	D0DSP — DECO 0 DBG_SHR_PTR	R	<a href="#">B.2.8.30/B-645</a>
0xBASE_8E18&	D0DMR — DECO 0 DBG_MID_REG	R	<a href="#">B.2.8.31/B-646</a>
0xBASE_8FA0 &#	CRNR — CHA Revision Number Register	R	<a href="#">B.2.5.1/B-535</a>
0xBASE_8FB0 #	SMSTA — Secure Memory Status Register	R	<a href="#">B.2.5.3/B-540</a>
0xBASE_8FB8 #	SMPO — Secure Memory Partition Owners Register	R	<a href="#">B.2.5.4/B-541</a>
0xBASE_8FB8 #	SMPO — Secure Memory Partition Owners Register	R	<a href="#">B.2.5.4/B-541</a>
0xBASE_8FC4 #	FAR — Fault Address Register	R	<a href="#">B.2.5.5/B-543</a>
0xBASE_8FC8 #	FAMR — Fault Address MID Register	R	<a href="#">/B-544</a>
0xBASE_8FCC #	FADR — Fault Address Detail Register	R	<a href="#">B.2.5.6/B-544</a>
0xBASE_8FD4 #	CSTA — CAAM Status Register	R	<a href="#">B.2.5.7/B-546</a>
0xBASE_8FD8 &#	SMVID — Secure Memory Version ID Register	R	<a href="#">B.2.5.8/B-548</a>
0xBASE_8FE4 #	CCBVID — CHA Cluster Block Version ID Register	R	<a href="#">B.2.5.9/B-549</a>
0xBASE_8FE8 &#	CHAVID — CHA Version ID Register	R	<a href="#">B.2.5.10/B-550</a>

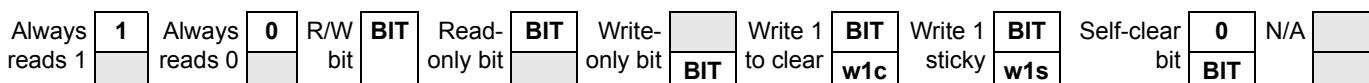
**Table B-9. CAAM Memory Map: DECO 0 and CCB 0**

Offset or Address	Register	Access	Section/Page
0xBASE_8FF0 &#	CHANUM — CHA Number Register	R	<a href="#">B.2.5.11/B-552</a>
0xBASE_8FF8 &#	CAAMVID — CAAM Version ID Register	R	<a href="#">B.2.5.12/B-554</a>

## B.1.2 Register Summary

A summary of the CAAM registers appears in [Table B-11](#). Full definitions of the CAAM registers appear in sections B.2.1 through B.2.8. Although many of the CAAM registers hold more than 32 bits, the Memory Map in [Section B.1.1 on page B-401](#), the Register Summary shown below, and the Register Definitions in the sections referenced above represent how these registers are accessed over the IP bus as 32-bit words. Only register words containing currently defined fields are represented in the register diagrams. The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. All addresses not shown are reserved. Note that multiple offset addresses followed by a single register name in parentheses indicate that the register is aliased to multiple addresses (i.e. the same register can be written or read using any of the addresses). Multiple offset addresses each of which is followed by a different register name indicate separate registers that use the same format (i.e. the field definitions are identical).

[Figure B-1](#) and [Table B-10](#) illustrate the register conventions used in describing the fields defined in these registers.

**Figure B-1. Key to Register Fields**

This table provides a key for register figures and tables. In order to maintain forward compatibility, nothing but 0 should ever be written to a register bit that is not writable in the current version of CAAM.

**Table B-10. Register Conventions**

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.

**Table B-10. Register Conventions**

Convention	Description
<b>Register Field Types</b>	
x	Read only. Writing this bit has no effect.
w	Write only.
rw	Standard read/write bit. Only software can change the bit's value (other than a hardware reset).
rwn	A read/write bit that may be modified by hardware in some fashion other than by a reset.
w1c	Write one to clear. A status bit that can be read, and is cleared by writing a one.
w1s	Write one sticky. (Sticky bit) Can be read. Resets to 0, and writing 0 has no effect. Writing 1 causes the bit to remain 1 until power on reset.
Self-clearing bit	Writing a one has some effect on the module, but it always reads as zero. (Previously designated slfclr)
Reset Values	
*	Resets to zero.
:	Resets to one.
-	Undefined at reset.
.	Unaffected by reset.
[signal_name]	Reset value is determined by polarity of indicated signal.

**Table B-11. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
<b>Configuration and Special Key Registers</b>																		
<b>Note:</b> Full definitions of these registers appear in <a href="#">Section B.2.1, “Configuration and Special Key Registers”</a> starting on page 461																		
0xBASE_0004 (MCFGR)	R	SWRST	WDE	WDF	0	0	0	0	0	0	0	0	0	0	0	0		
	W				DMARST													
	R	ARCACHE			AWCACHE				AXIPIPE				0	BURST				
	W																	

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0xBASE_000C (SCFGR)	R	MPCURVE				0	MPMRL														
	W																				
	R	0	0	0	0	0	RDB	RNGSH0	RANDDPAR	0	0	0	0	0	0	PRIBLOB					
	W							w1s	w1s							w1s	w1s				
0xBASE_0010 (JR0MIDR_MS) 0xBASE_0018 (JR1MIDR_MS)	R	LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AMTD				
	W																				
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JROWN					
	W																				
0xBASE_0014 (JR0MIDR_LS) 0xBASE_001C (JR1MIDR_LS)	R	0	0	0	0		JRNSMID														
	W																				
	R	0	0	0	0		JRSMID														
	W																				
0xBASE_009C (DECORR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DEN0				
	W																				
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RQD0				
	W																				
0xBASE_00A0 (DECO0MID_MS)	R	LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	W																				
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DOWN					
	W																				
0xBASE_00A4 (DECO0MID_LS)	R	0	0	0	0		DNSMID														
	W																				
	R	0	0	0	0		DSMID														
	W																				
0xBASE_0120 (DAR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	W																				
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NYA0				
	W																				

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0124 (DRR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																RST0
0xBASE_0400-041F (JDKEKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_0420-043F (TDKEKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_0440-045F (TDSKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_04E0 (SKNR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0													SK_NONCE_MS
	W																
0xBASE_04E4 (SKNR_LS)	R																SK_NONCE_LS
	W																
	R																SK_NONCE_LS
	W																
<b>RNG Registers</b>																	
Fields labeled with “**” are accessible only if RTMCTL register PRGM field is 1, else these fields read zeroes and are read-only																	
<b>Registers whose offset are followed by a “@x” subscript are accessible at that address only when RTMCTL register PRGM=x.</b>																	
<b>Note:</b> Full definitions of these registers appear in <a href="#">Section B.2.2, “RNG Registers”</a> starting on page 476																	

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
0xBASE_0600 (RTMCTL)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
	W															PRGM							
	R	0	0	TSTO_P_OK	ERR	TST_OUT	ENT_VAL	FCT_VAL	FCT_FAIL	FORCE_SYSCLK**	0	TRNG_ACC	CLK_OUT_EN	OSC_DIV**	SAMP_MODE**								
	W				CLR_ERR						RST_DEF**												
0xBASE_0604 (RTSCMISC)	R	0	0	0	0	0	0	0	0	0	0	0	0	RTY_CNT**									
	W																						
	R	0	0	0	0	0	0	0	0	LRUN_MAX**													
	W																						
0xBASE_0608 (RTPKRRNG)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
	W																						
	R	PKR_RNG**																					
	W																						
0xBASE_060C@1 (RTPKRMAX)	R	0	0	0	0	0	0	0	0	PKR_MAX	PKR_MAX												
	W																						
	R	PKR_MAX																					
	W																						
0xBASE_060C@0 (RTPKRSQ)	R	0	0	0	0	0	0	0	0	PKR_SQ													
	W																						
	R	PKR_SQ																					
	W																						
0xBASE_0610 (RTSDCTL)	R	ENT_DLY**																					
	W																						
	R	SAMP_SIZE**																					
	W																						
0xBASE_0614@1 (RTSBLIM)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
	W																						
	R	0	0	0	0	0	0	SB_LIM															
	W																						

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0614@0 (RTTOTSAM)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TOT_SAM
	W															
	R	TOT_SAM														
	W															
0xBASE_0618 (RTFRQMIN)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRQ_MIN**
	W															
	R	FRQ_MIN**														
	W															
0xBASE_061C@1 (RTFRQMAX)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRQ_MAX
	W															
	R	FRQ_MAX														
	W															
0xBASE_061C@0 (RTFRQCNT)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRQ_CNT
	W															
	R	FRQ_CNT														
	W															
0xBASE_0620@1 (RTSCML)	R	MONO_RNG														
	W															
	R	MONO_MAX														
	W															
0xBASE_0620@0 (RTSCMC)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	MONO_CNT														
	W															
0xBASE_0624@1 (RTSCR1L)	R	0	RUN1_RNG													
	W															
	R	0	RUN1_MAX													
	W															
0xBASE_0624@0 (RTSCR1C)	R	0	R1_1_COUNT													
	W															
	R	0	R1_0_COUNT													
	W															

**Table B-11. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0xBASE_0628@1 (RTSCR2L)	R	0	0	RUN2_RNG																
	W																			
	R	0	0	RUN2_MAX																
	W																			
0xBASE_0628@0 (RTSCR2C)	R	0	0	R2_1_COUNT																
	W																			
	R	0	0	R2_0_COUNT																
	W																			
0xBASE_062C@1 (RTSCR3L)	R	0	0	0	RUN3_RNG															
	W																			
	R	0	0	0	RUN3_MAX															
	W																			
0xBASE_062C@0 (RTSCR3C)	R	0	0	0	R3_1_COUNT															
	W																			
	R	0	0	0	R3_0_COUNT															
	W																			
0xBASE_0630@1 (RTSCR4L)	R	0	0	0			0	RUN4_RNG												
	W																			
	R	0	0	0			0	RUN4_MAX												
	W																			
0xBASE_0630@0 (RTSCR4C)	R	0	0	0			0	R4_1_COUNT												
	W																			
	R	0	0	0			0	R4_0_COUNT												
	W																			
0xBASE_0634@1 (RTSCR5L)	R	0	0	0			0	0	RUN5_RNG											
	W																			
	R	0	0	0			0	0	RUN5_MAX											
	W																			

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0634@0 (RTSCR5C)	R	0	0	0	0	0											R5_1_COUNT
	W																
	R	0	0	0	0	0											R5_0_COUNT
	W																
0xBASE_0638@1 (RTSCR6PL)	R	0	0	0	0	0											RUN6P_RNG
	W																
	R	0	0	0	0	0											RUN6P_MAX
	W																
0xBASE_0638@0 (RTSCR6PC)	R	0	0	0	0	0											R6P_1_COUNT
	W																
	R	0	0	0	0	0											R6P_0_COUNT
	W																
0xBASE_0640 (RTENT0)	R																ENT (TRNG entropy bits [383:368])**
	W																
	R																ENT (TRNG entropy bits [367:352])**
	W																
0xBASE_0644 (RTENT1)	R																ENT (TRNG entropy bits [351:336])**
	W																
	R																ENT (TRNG entropy bits [335:320])**
	W																
0xBASE_0648 (RTENT2)	R																ENT (TRNG entropy bits [319:304])**
	W																
	R																ENT (TRNG entropy bits [303:288])**
	W																
0xBASE_064C (RTENT3)	R																ENT (TRNG entropy bits [287:272])**
	W																
	R																ENT (TRNG entropy bits [271:256])**
	W																
0xBASE_0650 (RTENT4)	R																ENT (TRNG entropy bits [255:240])**
	W																
	R																ENT (TRNG entropy bits [239:224])**
	W																

**Table B-11. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0654 (RTENT5)	R	ENT (TRNG entropy bits [223:208])**														
	W															
	R	ENT (TRNG entropy bits [207:192])**														
	W															
0xBASE_0658 (RTENT6)	R	ENT (TRNG entropy bits [191:176])**														
	W															
	R	ENT (TRNG entropy bits [175:160])**														
	W															
0xBASE_065C (RTENT7)	R	ENT (TRNG entropy bits [159:144])**														
	W															
	R	ENT (TRNG entropy bits [143:128])**														
	W															
0xBASE_0660 (RTENT8)	R	ENT (TRNG entropy bits [127:112])**														
	W															
	R	ENT (TRNG entropy bits [111:96])**														
	W															
0xBASE_0664 (RTENT9)	R	ENT (TRNG entropy bits [95:80])**														
	W															
	R	ENT (TRNG entropy bits [79:64])**														
	W															
0xBASE_0668 (RTENT10)	R	ENT (TRNG entropy bits [63:48])**														
	W															
	R	ENT (TRNG entropy bits [47:32])**														
	W															
0xBASE_066C (RTENT11)	R	ENT (TRNG entropy bits [31:16])**														
	W															
	R	ENT (TRNG entropy bits [15:0])**														
	W															

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0680 (RTPKRCNT10)	R															PKR_1_CNT**
	W															
	R															PKR_0_CNT**
	W															
0xBASE_0684 (RTPKRCNT32)	R															PKR_3_CNT**
	W															
	R															PKR_2_CNT**
	W															
0xBASE_0688 (RTPKRCNT54)	R															PKR_5_CNT**
	W															
	R															PKR_4_CNT**
	W															
0xBASE_068C (RTPKRCNT76)	R															PKR_7_CNT**
	W															
	R															PKR_6_CNT**
	W															
0xBASE_0690 (RTPKRCNT98)	R															PKR_9_CNT**
	W															
	R															PKR_8_CNT**
	W															
0xBASE_0694 (RTPKRCNTBA)	R															PKR_B_CNT**
	W															
	R															PKR_A_CNT**
	W															
0xBASE_0698 (RTPKRCNTDC)	R															PKR_D_CNT**
	W															
	R															PKR_C_CNT**
	W															
0xBASE_069C (RTPKRCNTFE)	R															PKR_F_CNT**
	W															
	R															PKR_E_CNT**
	W															

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_06C0 (RDSTA)	R	SKVT	SKVN	0	0	0	0	0	0	0	0	CE	ERR CODE			
	W															
	R	0	0	0	0	TF3	TF2	TF1	TF0	PR3	PR2	PR1	PR0	IF3	IF2	IF1
	W															
0xBASE_06D0 (RDINT0)	R															
	W															
	R															
	W															
0xBASE_06D4 (RDINT1)	R															
	W															
	R															
	W															
0xBASE_06E0 (RDHCNTL)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HD
	W												HTM	0	0	
0xBASE_06E4 (RDHMDIG)	R															
	W															
	R															
	W															
0xBASE_06E8 (RDHBUF)	R											0	0	0	0	0
	W															
	R											0	0	0	0	0
	W															
<b>RNG Registers</b>																
<b>Note:</b> Full definitions of these registers appear in <a href="#">Section B.2.2, “RNG Registers”</a> starting on page 476																

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0BF8 0xBASE_0FF8 0xBASE_1FF8 0xBASE_2FF8 0xBASE_8FF8  (All aliases for CAAMVID_MS)	R															IP_ID
	W															
	R															MAJ_REV
	W															MIN_REV
0xBASE_0BFC 0xBASE_0FFC 0xBASE_1FFC 0xBASE_2FFC 0xBASE_8FFC  (All aliases for CAAMVID_LS)	R															COMPILE_OPT
	W															
	R															INTG_OPT
	W															
																ECO_REV
																CONFIG_OPT

**Job Queue Controller Debugging Registers**

**Note:** Full definitions of these registers appear in [Section B.2.3, “Job Queue Controller Debugging Registers”](#) starting on page 514.

0xBASE_0C00 (HT0_JD_ADDR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0								Reserved
	W															
0xBASE_0C04 (HT0_JD_ADDR_LS)	R															JD_ADDR
	W															
	R															JD_ADDR
	W															
0xBASE_0C08 (HT0_SD_ADDR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0								Reserved
	W															
0xBASE_0C0C (HT0_SD_ADDR_LS)	R															SD_ADDR
	W															
	R															SD_ADDR
	W															
0xBASE_0C10 (HT0_JQCTRL_MS)	R	0	0	WHL	FOUR	ILE			SHR_FROM	0	0	0	0	0	0	0
	W															
	R	AMTD		0	0	0			SRC	0	0	0	0			ID
	W															

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0C14 (HT0_JQCTRL_LS)	R	0	0	0	0	NON_SEQ_MID										
	W															
	R	0	0	0	0	SEQ_MID										
	W															
0xBASE_0C1C (HT0_STATUS)	R	BC	IN_USE	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PEND_0
	W															
0xBASE_0D70 (JRAAA0_MS) 0xBASE_0D78 (JRAAA1_MS) 0xBASE_0D80 (JRAAA2_MS) 0xBASE_0D88 (JRAAA3_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	Reserved			Reserved			
	W															
0xBASE_0D74 (JRAAA0_LS) 0xBASE_0D7C (JRAAA1_LS) 0xBASE_0D84 (JRAAA2_LS) 0xBASE_0D8C (JRAAA3_LS)	R	JD_ADDR														
	W															
	R	JD_ADDR														
	W															
0xBASE_0DB4 (JRAAVS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	V3	0	SRC3	V2	0	SRC2	V1	0	SRC1	V0	0	SRC0			
	W															
0xBASE_0DB8 (JRJIDU_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	JID15	JID14	JID13	JID12	JID11	JID10	JID09	JID08	JID07	JID06	JID05	JID04	JID03	JID02	JID01
	W															JID00

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0DBC (JRJIDU_LS)	R	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
0xBASE_0DC0 (JRJDJIF0)	R	0	0	0	0	JOB_ID_ENTRY0				0	0	0	0	JOB_ID_ENTRY1			
	W																
	R	0	0	0	0	JOB_ID_ENTRY2				0	0	0	0	JOB_ID_ENTRY3			
	W																
0xBASE_0DC4 (JRJDJIF1)	R	0	0	0	0	JOB_ID_ENTRY4				0	0	0	0	JOB_ID_ENTRY5			
	W																
	R	0	0	0	0	JOB_ID_ENTRY6				0	0	0	0	JOB_ID_ENTRY7			
	W																
0xBASE_0DC8 (JRJDJIF2)	R	0	0	0	0	JOB_ID_ENTRY8				0	0	0	0	JOB_ID_ENTRY9			
	W																
	R	0	0	0	0	JOB_ID_ENTRY10				0	0	0	0	JOB_ID_ENTRY11			
	W																
0xBASE_0DCC (JRJDJIF3)	R	0	0	0	0	JOB_ID_ENTRY12				0	0	0	0	JOB_ID_ENTRY13			
	W																
	R	0	0	0	0	JOB_ID_ENTRY14				0	0	0	0	JOB_ID_ENTRY15			
	W																
0xBASE_0DE0 (JRJDSF0)	R	0	0	SRC0		0	0	SRC1		0	0	SRC2		0	0	SRC3	
	W																
	R	0	0	SRC4		0	0	SRC5		0	0	SRC6		0	0	SRC7	
	W																
0xBASE_0DE4 (JRJDSF1)	R	0	0	SRC80		0	0	SRC9		0	0	SRC10		0	0	SRC11	
	W																
	R	0	0	SRC12		0	0	SRC13		0	0	SRC14		0	0	SRC15	
	W																
0xBASE_0DF0 (JRJDVF)	R	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																

**Table B-11. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0DF4 (JRJDS)	R	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
0xBASE_0E00 0xBASE_0E08 0xBASE_0E10 0xBASE_0E18 0xBASE_0E20 0xBASE_0E28 0xBASE_0E30 0xBASE_0E38 0xBASE_0E40 0xBASE_0E48 0xBASE_0E50 0xBASE_0E58 0xBASE_0E60 0xBASE_0E68 0xBASE_0E70 0xBASE_0E78	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	Reserved				Reserved		
	W															

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0E04 (0xBASE_0E0C (0xBASE_0E14 (0xBASE_0E1C (0xBASE_0E24 (0xBASE_0E2C (0xBASE_0E34 (0xBASE_0E3C (0xBASE_0E44 (0xBASE_0E4C (0xBASE_0E54 (0xBASE_0E5C (0xBASE_0E64 (0xBASE_0E6C (0xBASE_0E74 (0xBASE_0E7C (	R															
	W															
	R															
	W															
	R															
	W															
	R															
	W															

**Version ID Registers**

**Note:** Full definitions of these registers appear in the sections beginning with  
[Section B.2.5.1, “CHA Revision Number Register \(CRNR\)” starting on page 535](#)

0xBASE_0FA0 0xBASE_1FA0 0xBASE_2FA0 0xBASE_8FA0 (All aliases for CRNR_MS)	R	JRRN				DECORN				0	0	0	0	0	0	0
	W															
	R	0				0				0				0		
	W															
0xBASE_0FA4 0xBASE_1FA4 0xBASE_2FA4 0xBASE_8FA4 (All aliases for CRNR_LS)	R	0				0				0				RNGRN		
	W															
	R	MDRN				ARC4RN				DESRN				AESRN		
	W															

**Table B-11. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

**Secure Memory Status Registers**

**Note:** Full definitions of these registers appear in the sections beginning with  
[Section B.2.5.3, “Secure Memory Status Register \(SMSTA\)” starting on page 540](#)

0xBASE_0FB4 0xBASE_1FB4 0xBASE_2FB4 0xBASE_8FB4 (All aliases for SMSTA)	R	PART			0	PAGE											
	W																
	R	0	0	MID				ACCERR				STATE					
	W																
0xBASE_1FBC 0xBASE_2FBC (All aliases for SMPO)	R	PO15		PO14		PO13		PO12		PO11		PO10		PO9		PO8	
	W																
	R	PO7		PO6		PO5		PO4		PO3		PO2		PO1		PO0	
	W																

**Bus Fault Registers**

**Note:** Full definitions of these registers appear in the sections beginning with  
[Section B.2.5.5, “Fault Address Register \(FAR\)” starting on page 543](#)

0xBASE_0FC4 0xBASE_1FC4 0xBASE_2FC4 0xBASE_8FC4 (All aliases for FAR)	R	FAR														
	W															
	R	FAR														
	W															
0xBASE_0FC8 0xBASE_1FC8 0xBASE_2FC8 0xBASE_8FC8 (All aliases for FAMR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	FMID														
	W															
0xBASE_0FCC 0xBASE_1FCC 0xBASE_2FCC 0xBASE_8FCC (All aliases for FADR)	R	FERR		0	0	0	0	0	0	0	0	0	0	0	0	FSZ_EXT
	W															
	R	DTYP	JSRC			BLKID				TYP	FSZ					
	W															

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
<b>CAAM Status and Version ID Registers</b>																	
<b>Note:</b> Full definitions of these registers appear in the sections beginning with <a href="#">Section B.2.5.7, “CAAM Status Register (CSTA)”</a> starting on page 546																	
0xBASE_0FD4 0xBASE_1FD4 0xBASE_2FD4 0xBASE_8FD4 (All aliases for CSTA)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	0	0	0	0	0	PLEND	MOO		0	0	0	0	0	TRNG_IDLE	IDLE	
	W															BSY	
0xBASE_0FD8 0xBASE_1FD8 0xBASE_2FD8 0xBASE_8FD8 (All aliases for SMVID_MS)	R	0	0	0	0	0	0	MAX_NPAG									
	W																
	R	NPRT			0	0	NPAG										
	W																
0xBASE_0FDC 0xBASE_1FDC 0xBASE_2FDC 0xBASE_8FDC (All aliases for SMVID_LS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	PSIZ		
	W																
	R	SMJV						SMNV									
	W																
0xBASE_0FE0 0xBASE_1FE0 0xBASE_2FE0 0xBASE_8FE0 (All aliases for RVID)	R	TZ	0	0	0	MD	MC	MB	MA	0	0	0	MBL	0	0	SHA-256	
	W															0	
	R	RMJV								RMNV							
	W																
0xBASE_0FE4 0xBASE_1FE4 0xBASE_2FE4 0xBASE_8FE4 (All aliases for CCBVID)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	AMJV								AMNV							
	W																

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0FE8 0xBASE_1FE8 0xBASE_2FE8 0xBASE_8FE8 (All aliases for CHAVID_MS)	R	JRVID				DECVID				0	0	0	0	0	0	0
	W															
	R	0				0				0				0		
	W															
0xBASE_0FEC 0xBASE_1FEC 0xBASE_2FEC 0xBASE_8FEC (All aliases for CHAVID_LS)	R	0				0				0				RNGVID		
	W															
	R	MDVID				ARC4VID				DESVID				AESVID		
	W															
0xBASE_0FF0 0xBASE_1FF0 0xBASE_2FF0 0xBASE_8FF0 (All aliases for CHANUM_MS)	R	JRNUM				DECONUM				0	0	0	0	0	0	0
	W															
	R	0				0				0				0		
	W															
0xBASE_0FF4 0xBASE_1FF4 0xBASE_2FF4 0xBASE_8FF4 (CHANUM_LS)	R	0				0				0				RNGNUM		
	W															
	R	MDNUM				ARC4NUM				DESENUM				AESNUM		
	W															

**Job Ring Registers**

**Note:** Full definitions of these registers appear in [Section B.2.6, “Job Ring Registers](#) starting on page 556

0xBASE_1004 (IRBAR0) 0xBASE_2004 (IRBAR1)	R	IRBA														
	W															
	R	IRBA														
	W															
0xBASE_100C (IRSR0) 0xBASE_200C (IRSR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	IRS								
	W															
0xBASE_1014 (IRSAR0) 0xBASE_2014 (IRSAR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	IRSA								
	W															

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_101C (IRJAR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	IRJA									
	W																
0xBASE_1024 (ORBAR0)	R	ORBA															
	W																
	R	ORBA															
	W																
0xBASE_102C (ORSR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	ORS									
	W																
0xBASE_1034 (ORJRR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	ORJR									
	W																
0xBASE_103C (ORSFR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	ORSF									
	W																
0xBASE_1044 (JRSTAR0)	R	SSRC				SSED											
	W																
	R	SSED															
	W																
0xBASE_104C (JRINTR0)	R	0	0	ERR_ORWI													
	W																
	R	0	0	0	0	ERR_TYPE				0	0	0	0	HALT		JRE	JRI
	W														w1c	w1c	
0xBASE_1050 (JRCFG0_MS)	R	0	0	0	0												
	W	JDMS															
	R		RSV	RSV	CHWSO	CBSO	RSV	RSV	MHWSO	MBSO	RSV	RSV	CHWSI	CBSI	RSV	RSV	MHSWI
	W																DMBS

Table B-11. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_1054 (JRCFGR0_LS) 0xBASE_2054 (JRCFGR1_LS)	R	ICTT														
	W															
	R	ICDCT								0	0	0	0	0	0	ICEN
	W															IMSK
0xBASE_105C (IRRIR0) 0xBASE_205C (IRRIR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	IRRRI											
	W															
0xBASE_1064 (ORWIR0) 0xBASE_2064 (ORWIR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	ORWI												
	W															
0xBASE_106C (JRCR0) 0xBASE_206C (JRCR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
<b>Secure Memory Registers</b>																

**Note:** Full definitions of these registers appear in [Section B.2.7, “Secure Memory Registers](#) starting on page 582

0xBASE_10F4 (SMCJR0) 0xBASE_20F4 (SMCJR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W	PAGE															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W									PRTN			CMD				
0xBASE_10FC (SMCSJR0) 0xBASE_20FC (SMCSJR1)	R	0	0	0	0	PAGE											
	W																
	R	CERR	AERR		0	0	0	0	PO		0	0	PRTN				
	W																
0xBASE_11X4 0xBASE_21X4 (All aliases for SMAPJR <sub>x</sub> )	R	PARTITION_KMOD															
	W																
	R	CSP	PSP	SMAP_LCK	SMAG	JR		0	0	G2_SMBLOB	1	G2_WRITE	G2_READ	G1_SMBLOB	1	G1_WRITE	G1_READ
	W				LCK												

Table B-11. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_11X8 0xBASE_21X8 (All aliases for SMAG2x)	R	G2_31	G2_30	G2_29	G2_28	G2_27	G2_26	G2_25	G2_24	G2_23	G2_22	G2_21	G2_20	G2_19	G2_18	G2_17	G2_16
	W	G2_15	G2_14	G2_13	G2_12	G2_11	G2_10	G2_9	G2_8	G2_7	G2_6	G2_5	G2_4	G2_3	G2_2	G2_1	G2_0
	R	G1_31	G1_30	G1_29	G1_28	G1_27	G1_26	G1_25	G1_24	G1_23	G1_22	G1_21	G1_20	G1_19	G1_18	G1_17	G1_16
	W	G1_15	G1_14	G1_13	G1_12	G1_11	G1_10	G1_9	G1_8	G1_7	G1_6	G1_5	G1_4	G1_3	G1_2	G1_1	G1_0

### B.1.3 Register Summary

A summary of the CAAM registers appears in [Table B-11](#). Full definitions of the CAAM registers appear in sections B.2.1 through B.2.8. Although many of the CAAM registers hold more than 32 bits, the Memory Map in section [B.1.1 on page B-401](#), the Register Summary shown below, and the Register Definitions in the sections referenced above represent how these registers are accessed over the IP bus as 32-bit words. Only register words containing currently defined fields are represented in the register diagrams. The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. All addresses not shown are reserved. Note that multiple offset addresses followed by a single register name in parentheses indicate that the register is aliased to multiple addresses (i.e. the same register can be written or read using any of the addresses). Multiple offset addresses each of which is followed by a different register name indicate separate registers that use the same format (i.e. the field definitions are identical).

[Figure B-1](#) and [Table B-10](#) illustrate the register conventions used in describing the fields defined in these registers.

Always reads 1		Always reads 0		R/W bit		Read-only bit		Write-only bit		Write 1 to clear		Write 1 sticky		Self-clear bit		N/A	
----------------	--	----------------	--	---------	--	---------------	--	----------------	--	------------------	--	----------------	--	----------------	--	-----	--

Figure B-2. Key to Register Fields

This table provides a key for register figures and tables. In order to maintain forward compatibility, nothing but 0 should ever be written to a register bit that is not writable in the current version of CAAM.

**Table B-12. Register Conventions**

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
fieldname	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
<b>Register Field Types</b>	
r	Read only. Writing this bit has no effect.
w	Write only.
rw	Standard read/write bit. Only software can change the bit's value (other than a hardware reset).
ro	A read/write bit that may be modified by hardware in some fashion other than by a reset.
w1c	Write one to clear. A status bit that can be read, and is cleared by writing a one.
w1s	Write one sticky. (Sticky bit) Can be read. Resets to 0, and writing 0 has no effect. Writing 1 causes the bit to remain 1 until power on reset.
self-clearing bit	Writing a one has some effect on the module, but it always reads as zero. (Previously designated slfclr)
Reset Values	
s	Resets to zero.
t	Resets to one.
—	Undefined at reset.
—	Unaffected by reset.
[signal_name]	Reset value is determined by polarity of indicated signal.

**Table B-13. CAAM Register Summary**

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>Configuration and Special Key Registers</b>																
<b>Note:</b> Full definitions of these registers appear in <a href="#">Section B.2.1, “Configuration and Special Key Registers”</a> starting on page 461																
0xBASE_0004 (MCFGGR)	R W	SWRST	WDE	WDF	0 DMARST	0	0	0	0	0	0 ○	0 ○	0	0	0	0
	R W	ARCACHE			AWCACHE				AXIPIPE				0	BURST		

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0xBASE_000C (SCFGR)	R	MPCURVE				0	MPMRL													
	W																			
	R	0	0	0	0	0	RDB	RNGSH0	RANDDPAR	0	0	0	0	0	0	PRIBLOB				
	W						w1s	w1s	w1s							w1s	w1s			
0xBASE_0010 (JR0MIDR_MS) 0xBASE_0018 (JR1MIDR_MS)	R	LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AMTD			
	W																			
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JROWN				
	W																			
0xBASE_0014 (JR0MIDR_LS) 0xBASE_001C (JR1MIDR_LS)	R	0	0	0	0	JRNSMID														
	W																			
	R	0	0	0	0	JRSMID														
	W																			
0xBASE_009C (DECORR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DEN0			
	W																			
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RQD0			
	W																			
0xBASE_00A0 (DECO0MID_MS)	R	LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	W																			
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DOWN				
	W																			
0xBASE_00A4 (DECO0MID_LS)	R	0	0	0	0	DNSMID														
	W																			
	R	0	0	0	0	DSMID														
	W																			
0xBASE_0120 (DAR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	W																			
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NYA0			
	W																			

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0124 (DRR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																RST0
0xBASE_0400-041F (JDKEKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_0420-043F (TDKEKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_0440-045F (TDSKR)	R																
	W																bits 255-0 (R/W in non-secure mode only)
	R																
	W																
0xBASE_04E0 (SKNR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	SK_NONCE_MS												
	W																
0xBASE_04E4 (SKNR_LS)	R				SK_NONCE_LS												
	W																
	R																
	W				SK_NONCE_LS												

**RNG Registers**

Fields labeled with “\*\*” are accessible only if RTMCTL register PRGM field is 1, else these fields read zeroes and are read-only.  
**Registers whose offset are followed by a “@x” subscript are accessible at that address only when RTMCTL register PRGM=x.**

**Note:** Full definitions of these registers appear in [Section B.2.2, “RNG Registers”](#) starting on page page 476

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16														
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
0xBASE_0600 (RTMCTL)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREGM														
	W																														
	R	0	0	TSTO_P_OK	ERR	TST_OUT	ENT_VAL	FCT_VAL	FCT_FAIL	FORCE_SYSCLK**	0	RST_DEF**	TRNG_ACC	CLK_OUT_EN	OSC_DIV**	SAMP_MODE**															
	W				CLR_ERR																										
0xBASE_0604 (RTSCMISC)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RTY_CNT**														
	W																														
	R	0	0	0	0	0	0	0	0	LRUN_MAX**																					
	W																														
0xBASE_0608 (RTPKRRNG)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														
	W																														
	R	PKR_RNG**																													
	W																														
0xBASE_060C@1 (RTPKRMAX)	R	0	0	0	0	0	0	0	0	PKR_MAX																					
	W																														
	R	PKR_MAX																													
	W																														
0xBASE_060C@0 (RTPKRSQ)	R	0	0	0	0	0	0	0	0	PKR_SQ																					
	W																														
	R	PKR_SQ																													
	W																														
0xBASE_0610 (RTSDCTL)	R	ENT_DLY**																													
	W																														
	R	SAMP_SIZE**																													
	W																														
0xBASE_0614@1 (RTSBLIM)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														
	W																														
	R	0	0	0	0	0	0	SB_LIM																							
	W																														

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0614@0 (RTTOTSAM)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TOT_SAM
	W																
	R	TOT_SAM															
	W																
0xBASE_0618 (RTFRQMIN)	R	0	0	0	0	0	0	0	0	0	0	0	FRQ_MIN**				
	W																
	R	FRQ_MIN**															
	W																
0xBASE_061C@1 (RTFRQMAX)	R	0	0	0	0	0	0	0	0	0	0	0	FRQ_MAX				
	W																
	R	FRQ_MAX															
	W																
0xBASE_061C@0 (RTFRQCNT)	R	0	0	0	0	0	0	0	0	0	0	0	FRQ_CNT				
	W																
	R	FRQ_CNT															
	W																
0xBASE_0620@1 (RTSCML)	R	MONO_RNG															
	W																
	R	MONO_MAX															
	W																
0xBASE_0620@0 (RTSCMC)	R	0	0	0	0	0	0	0	0	0	0	0	MONO_CNT				
	W																
	R	MONO_CNT															
	W																
0xBASE_0624@1 (RTSCR1L)	R	0	RUN1_RNG														
	W																
	R	0	RUN1_MAX														
	W																
0xBASE_0624@0 (RTSCR1C)	R	0	R1_1_COUNT														
	W																
	R	0	R1_0_COUNT														
	W																

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0628@1 (RTSCR2L)	R	0	0														RUN2_RNG
	W																
	R	0	0														RUN2_MAX
	W																
0xBASE_0628@0 (RTSCR2C)	R	0	0														R2_1_COUNT
	W																
	R	0	0														R2_0_COUNT
	W																
0xBASE_062C@1 (RTSCR3L)	R	0	0	0													RUN3_RNG
	W																
	R	0	0	0													RUN3_MAX
	W																
0xBASE_062C@0 (RTSCR3C)	R	0	0	0													R3_1_COUNT
	W																
	R	0	0	0													R3_0_COUNT
	W																
0xBASE_0630@1 (RTSCR4L)	R	0	0	0	0												RUN4_RNG
	W																
	R	0	0	0	0												RUN4_MAX
	W																
0xBASE_0630@0 (RTSCR4C)	R	0	0	0	0												R4_1_COUNT
	W																
	R	0	0	0	0												R4_0_COUNT
	W																
0xBASE_0634@1 (RTSCR5L)	R	0	0	0	0	0											RUN5_RNG
	W																
	R	0	0	0	0	0											RUN5_MAX
	W																
0xBASE_0634@0 (RTSCR5C)	R	0	0	0	0	0											R5_1_COUNT
	W																
	R	0	0	0	0	0											R5_0_COUNT
	W																

**Table B-13. CAAM Register Summary**

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0638@1 (RTSCR6PL)	R	0	0	0	0	0	RUN6P_RNG										
	W																
	R	0	0	0	0	0	RUN6P_MAX										
	W																
0xBASE_0638@0 (RTSCR6PC)	R	0	0	0	0	0	R6P_1_COUNT										
	W																
	R	0	0	0	0	0	R6P_0_COUNT										
	W																
0xBASE_0640 (RTENT0)	R	ENT (TRNG entropy bits [383:368])**															
	W																
	R	ENT (TRNG entropy bits [367:352])**															
	W																
0xBASE_0644 (RTENT1)	R	ENT (TRNG entropy bits [351:336])**															
	W																
	R	ENT (TRNG entropy bits [335:320])**															
	W																
0xBASE_0648 (RTENT2)	R	ENT (TRNG entropy bits [319:304])**															
	W																
	R	ENT (TRNG entropy bits [303:288])**															
	W																
0xBASE_064C (RTENT3)	R	ENT (TRNG entropy bits [287:272])**															
	W																
	R	ENT (TRNG entropy bits [271:256])**															
	W																
0xBASE_0650 (RTENT4)	R	ENT (TRNG entropy bits [255:240])**															
	W																
	R	ENT (TRNG entropy bits [239:224])**															
	W																
0xBASE_0654 (RTENT5)	R	ENT (TRNG entropy bits [223:208])**															
	W																
	R	ENT (TRNG entropy bits [207:192])**															
	W																

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0658 (RTENT6)	R	ENT (TRNG entropy bits [191:176])**														
	W															
	R	ENT (TRNG entropy bits [175:160])**														
	W															
0xBASE_065C (RTENT7)	R	ENT (TRNG entropy bits [159:144])**														
	W															
	R	ENT (TRNG entropy bits [143:128])**														
	W															
0xBASE_0660 (RTENT8)	R	ENT (TRNG entropy bits [127:112])**														
	W															
	R	ENT (TRNG entropy bits [111:96])**														
	W															
0xBASE_0664 (RTENT9)	R	ENT (TRNG entropy bits [95:80])**														
	W															
	R	ENT (TRNG entropy bits [79:64])**														
	W															
0xBASE_0668 (RTENT10)	R	ENT (TRNG entropy bits [63:48])**														
	W															
	R	ENT (TRNG entropy bits [47:32])**														
	W															
0xBASE_066C (RTENT11)	R	ENT (TRNG entropy bits [31:16])**														
	W															
	R	ENT (TRNG entropy bits [15:0])**														
	W															
0xBASE_0680 (RTPKRCNT10)	R	PKR_1_CNT**														
	W															
	R	PKR_0_CNT**														
	W															
0xBASE_0684 (RTPKRCNT32)	R	PKR_3_CNT**														
	W															
	R	PKR_2_CNT**														
	W															

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0688 (RTPKRCNT54)	R	PKR_5_CNT**															
	W																
	R	PKR_4_CNT**															
	W																
0xBASE_068C (RTPKRCNT76)	R	PKR_7_CNT**															
	W																
	R	PKR_6_CNT**															
	W																
0xBASE_0690 (RTPKRCNT98)	R	PKR_9_CNT**															
	W																
	R	PKR_8_CNT**															
	W																
0xBASE_0694 (RTPKRCNTBA)	R	PKR_B_CNT**															
	W																
	R	PKR_A_CNT**															
	W																
0xBASE_0698 (RTPKRCNTDC)	R	PKR_D_CNT**															
	W																
	R	PKR_C_CNT**															
	W																
0xBASE_069C (RTPKRCNTFE)	R	PKR_F_CNT**															
	W																
	R	PKR_E_CNT**															
	W																
0xBASE_06C0 (RDSTA)	R	SKVT	SKVN	0	0	0	0	0	0	0	0	0	CE	ERR CODE			
	W																
	R	0	0	0	0	TF3	TF2	TF1	TF0	PR3	PR2	PR1	PR0	IF3	IF2	IF1	IFO
	W																

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_06D0 (RDINT0)	R															
	W															
	R															
	W															
0xBASE_06D4 (RDINT1)	R															
	W															
	R															
	W															
0xBASE_06E0 (RDHCNTL)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	HTM	0	0
	W														HI	HB
0xBASE_06E4 (RDHDIG)	R															
	W															
	R															
	W															
0xBASE_06E8 (RDHBUF)	R												0	0	0	0
	W															
	R												0	0	0	0
	W															

**RNG Registers**

**Note:** Full definitions of these registers appear in [Section B.2.2, “RNG Registers”](#) starting on page page 476

0xBASE_0BF8 0xBASE_0FF8 0xBASE_1FF8 0xBASE_2FF8 0xBASE_8FF8  (All aliases for CAAMVID_MS)	R																IP_ID
	W																
	R																MIN_REV
	W																
	R																COMPILER_OPT
0xBASE_0BFC 0xBASE_0FFC 0xBASE_1FFC 0xBASE_2FFC 0xBASE_8FFC  (All aliases for CAAMVID_LS)	R																INTG_OPT
	W																
	R																ECO_REV
	W																CONFIG_OPT

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>Job Queue Controller Debugging Registers</b>																
<b>Note:</b> Full definitions of these registers appear in <a href="#">Section B.2.3, “Job Queue Controller Debugging Registers”</a> starting on page 514																
0xBASE_0C00 (HT0_JD_ADDR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0							Reserved
	W															
0xBASE_0C04 (HT0_JD_ADDR_LS)	R	JD_ADDR														
	W															
	R	JD_ADDR														
	W															
0xBASE_0C08 (HT0_SD_ADDR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0							Reserved
	W															
0xBASE_0C0C (HT0_SD_ADDR_LS)	R	SD_ADDR														
	W															
	R	SD_ADDR														
	W															
0xBASE_0C10 (HT0_JQCTRL_MS)	R	0	0	WHL	FOUR	ILE	SHR_FROM			0	0	0	0	0	0	0
	W															
	R	AMTD		0	0	0	SRC			0	0	0	0	ID		
	W															
0xBASE_0C14 (HT0_JQCTRL_LS)	R	0	0	0	0	NON_SEQ_MID										
	W															
	R	0	0	0	0	SEQ_MID										
	W															

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0C1C (HT0_STATUS)	R	BC	IN_SS_	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PEND_Q
	W																
0xBASE_0D70 (JRAAA0_MS) 0xBASE_0D78 (JRAAA1_MS) 0xBASE_0D80 (JRAAA2_MS) 0xBASE_0D88 (JRAAA3_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	Reserved			Reserved				
	W																
0xBASE_0D74 (JRAAA0_LS) 0xBASE_0D7C (JRAAA1_LS) 0xBASE_0D84 (JRAAA2_LS) 0xBASE_0D8C (JRAAA3_LS)	R	JD_ADDR															
	W																
	R	JD_ADDR															
	W																
0xBASE_0DB4 (JRRAVS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	V3	0	SRC3	V2	0	SRC2	V1	0	SRC1	V0	0	SRC0				
	W																
0xBASE_0DB8 (JRJIDU_MS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	JID15	JID14	JID13	JID12	JID11	JID10	JID09	JID08	JID07	JID06	JID05	JID04	JID03	JID02	JID01	JID00
	W																
0xBASE_0DBC (JRJIDU_LS)	R	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
0xBASE_0DC0 (JRJDJIF0)	R	0	0	0	0	JOB_ID_ENTRY0				0	0	0	0	JOB_ID_ENTRY1			
	W																
	R	0	0	0	0	JOB_ID_ENTRY2				0	0	0	0	JOB_ID_ENTRY3			
	W																

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0DC4 (JRJDJIF1)	R	0	0	0	0	JOB_ID_ENTRY4				0	0	0	0	JOB_ID_ENTRY5			
	W																
	R	0	0	0	0	JOB_ID_ENTRY6				0	0	0	0	JOB_ID_ENTRY7			
	W																
0xBASE_0DC8 (JRJDJIF2)	R	0	0	0	0	JOB_ID_ENTRY8				0	0	0	0	JOB_ID_ENTRY9			
	W																
	R	0	0	0	0	JOB_ID_ENTRY10				0	0	0	0	JOB_ID_ENTRY11			
	W																
0xBASE_0DCC (JRJDJIF3)	R	0	0	0	0	JOB_ID_ENTRY12				0	0	0	0	JOB_ID_ENTRY13			
	W																
	R	0	0	0	0	JOB_ID_ENTRY14				0	0	0	0	JOB_ID_ENTRY15			
	W																
0xBASE_0DE0 (JRJDSF0)	R	0	0	SRC0		0	0	SRC1		0	0	SRC2		0	0	SRC3	
	W																
	R	0	0	SRC4		0	0	SRC5		0	0	SRC6		0	0	SRC7	
	W																
0xBASE_0DE4 (JRJDSF1)	R	0	0	SRC80		0	0	SRC9		0	0	SRC10		0	0	SRC11	
	W																
	R	0	0	SRC12		0	0	SRC13		0	0	SRC14		0	0	SRC15	
	W																
0xBASE_0DF0 (JRJDVF)	R	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
0xBASE_0DF4 (JRJDS)	R	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0E00 0xBASE_0E08 0xBASE_0E10 0xBASE_0E18 0xBASE_0E20 0xBASE_0E28 0xBASE_0E30 0xBASE_0E38 0xBASE_0E40 0xBASE_0E48 0xBASE_0E50 0xBASE_0E58 0xBASE_0E60 0xBASE_0E68 0xBASE_0E70 0xBASE_0E78	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	Reserved				Reserved		
	W															

**Table B-13. CAAM Register Summary**

## Version ID Registers

**Note:** Full definitions of these registers appear in the sections beginning with Section B.2.5.1, “CHA Revision Number Register (CRNR)” starting on page 535

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
<b>Secure Memory Status Registers</b>																	
<b>Note:</b> Full definitions of these registers appear in the sections beginning with <a href="#">Section B.2.5.3, "Secure Memory Status Register (SMSTA)"</a> starting on page page 540																	
0xBASE_0FB4 0xBASE_1FB4 0xBASE_2FB4 0xBASE_8FB4 (All aliases for SMSTA)	R	PART			0	PAGE											
	W																
	R	0	0	MID					ACCERR			STATE					
	W																
0xBASE_1FBC 0xBASE_2FBC (All aliases for SMPO)	R	PO15		PO14		PO13		PO12		PO11		PO10		PO9		PO8	
	W																
	R	PO7		PO6		PO5		PO4		PO3		PO2		PO1		PO0	
	W																
<b>Bus Fault Registers</b>																	
<b>Note:</b> Full definitions of these registers appear in the sections beginning with <a href="#">Section B.2.5.5, "Fault Address Register (FAR)"</a> starting on page page 543																	
0xBASE_0FC4 0xBASE_1FC4 0xBASE_2FC4 0xBASE_8FC4 (All aliases for FAR)	R	FAR															
	W																
	R	FAR															
	W																
0xBASE_0FC8 0xBASE_1FC8 0xBASE_2FC8 0xBASE_8FC8 (All aliases for FAMR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	FMID															
	W																
0xBASE_0FCC 0xBASE_1FCC 0xBASE_2FCC 0xBASE_8FCC (All aliases for FADR)	R	FERR		0	0	0	0	0	0	0	0	0	0	0	FSZ_EXT		
	W																
	R	D <sub>TYP</sub>	JSRC			BLKID				TYP	FSZ						
	W																
<b>CAAM Status and Version ID Registers</b>																	
<b>Note:</b> Full definitions of these registers appear in the sections beginning with <a href="#">Section B.2.5.7, "CAAM Status Register (CSTA)"</a> starting on page page 546																	

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xBASE_0FD4 0xBASE_1FD4 0xBASE_2FD4 0xBASE_8FD4 (All aliases for CSTA)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	PLEND	MOO		0	0	0	0	0	TRNG_IDLE	IDLE	BSY
	W																
0xBASE_0FD8 0xBASE_1FD8 0xBASE_2FD8 0xBASE_8FD8 (All aliases for SMVID_MS)	R	0	0	0	0	0	0	MAX_NPAG									
	W																
	R	NPRT			0	0	NPAG										
	W																
0xBASE_0FDC 0xBASE_1FDC 0xBASE_2FDC 0xBASE_8FDC (All aliases for SMVID_LS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	PSIZ		
	W																
	R	SMJV								SMNV							
	W																
(All aliases for RVID)	R	TZ	0	0	0	MD	MC	MB	MA	0	0	0	MBL	0	0	SHA-256	0
	W																
	R	RMJV								RMNV							
	W																
0xBASE_0FE4 0xBASE_1FE4 0xBASE_2FE4 0xBASE_8FE4 (All aliases for CCBVID)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	AMJV								AMNV							
	W																
0xBASE_0FE8 0xBASE_1FE8 0xBASE_2FE8 0xBASE_8FE8 (All aliases for CHAVID_MS)	R	JRVID			DECVID				0	0	0	0	0	0	0	0	0
	W																
	R	0				0				0				0			
	W																

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0xBASE_0FEC 0xBASE_1FEC 0xBASE_2FEC 0xBASE_8FEC (All aliases for CHAVID_LS)	R	0				0				0				RNGVID			
	W																
	R	MDVID				ARC4VID				DESVID				AESVID			
	W																
0xBASE_0FF0 0xBASE_1FF0 0xBASE_2FF0 0xBASE_8FF0 (All aliases for CHANUM_MS)	R	JRNUM				DECONUM				0	0	0	0	0	0	0	0
	W																
	R	0				0				0				0			
	W																
0xBASE_0FF4 0xBASE_1FF4 0xBASE_2FF4 0xBASE_8FF4 (CHANUM_LS)	R	0				0				0				RNGNUM			
	W																
	R	MDNUM				ARC4NUM				DESNUM				AESNUM			
	W																

**Job Ring Registers**

**Note:** Full definitions of these registers appear in [Section , “](#) starting on page page 555

0xBASE_1004 (IRBAR0_LS) 0xBASE_2004 (IRBAR1_LS)	R	IRBA														
	W															
	R	IRBA														
	W															
0xBASE_100C (IRSR0) 0xBASE_200C (IRSR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	IRS								
	W															
0xBASE_101C (IRJAR0) 0xBASE_201C (IRJAR1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	IRJA								
	W															
0xBASE_1024 (ORBAR0_LS) 0xBASE_2024 (ORBAR1_LS)	R	ORBA														
	W															
	R	ORBA														
	W															

Table B-13. CAAM Register Summary

Address (Register Name)		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0xBASE_102C (ORSR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																	
	R	0	0	0	0	0	0											
	W																	
0xBASE_1034 (ORJRR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																	
	R	0	0	0	0	0	0											
	W																	
0xBASE_1044 (JRSTAR0)	R	SSRC			SSED													
	W																	
	R	SSED																
	W																	
0xBASE_1050 (JRCFGRO_MS)	R	INCL SEQ OUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PEO	DMBS
	W																	
	R	RSV	RSV	CHWSO	CBSO	RSV	RSV	MHWSO	MBSO	RSV	RSV	CHWSI	CBSI	RSV	RSV	MHWSI	MBSI	
	W																	
0xBASE_1054 (JRCFGRO_LS)	R	ICTT																
	W																	
	R	ICDCT							0	0	0	0	0	0	0	ICEN	IMSK	
	W																	

## Secure Memory Registers

Note: Full definitions of these registers appear in [Section , “](#) starting on page [page 582](#)

0xBASE_10F4 (SMCJR0)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	W	PAGE																	
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	W	PRTN														CMD			
0xBASE_10FC (SMCSJR0)	R	0	0	0	0	PAGE													
	W																		
	R	CERR		AERR		0	0	0	0	PO		0	0	PRTN					
	W																		

Table B-13. CAAM Register Summary

Address (Register Name)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0xBASE_11X4 0xBASE_21X4 (All aliases for SMAPJRx)	R	PARTITION_KMOD															
	W																
	R	CSP	PSP	SMAP_LCK	SMAG_LCK	JR		0	0	G2_SMBLOB	1	G2_WRITE	G2_READ	G1_SMBLOB	1	G1_WRITE	G1_READ
	W																
0xBASE_11X8 0xBASE_21X8 (All aliases for SMAG2x)	R	G2_3	G2_3	G2_2	G2_2	G2_2	G2_2	G2_2	G2_2	G2_2	G2_2	G2_2	G2_1	G2_1	G2_1	G2_1	
	W	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6
	R	G2_1	G2_1	G2_1	G2_1	G2_1	G2_1	G2_9	G2_1	G2_7	G2_6	G2_5	G2_4	G2_3	G2_2	G2_1	G2_0
	W	5	4	3	2	1	0										
0xBASE_11XC 0xBASE_21XC (All aliases for SMAG1x)	R	G1_3	G1_3	G1_2	G1_2	G1_2	G1_2	G1_2	G1_2	G1_2	G1_2	G1_2	G1_1	G1_1	G1_1	G1_1	
	W	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6
	R	G1_1	G1_1	G1_1	G1_1	G1_1	G1_1	G1_9	G1_1	G1_7	G1_6	G1_5	G1_4	G1_3	G1_2	G1_1	G1_0
	W	5	4	3	2	1	0										

## DECO and CCB Registers

Note: Full definitions of these registers appear in [Section B.2.8, “DECO and CCB Registers](#) starting on page [page 597](#)

0xBASE_8004 (C0C1MR)	R	0	0	0	0	0	0	0	0	ALG						
	W															
	R	0	0	0	0	AAI							AS		ICV	ENC
	W															
0xBASE_800C (C0C1KSR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	C1KS						
	W															
0xBASE_8010 (C0C1DSR_MS)	R	NUMBITS			0	0	0	0	0	0	0	0	0	0	0	0
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C1CY
	W															
0xBASE_8014 (C0C1DSR_LS)	R	C1DS														
	W															
	R															
	W															

0xBASE_801C (C0C1ICVSR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																	
	R	0	0	0	0	0	0											
	W																	
0xBASE_8034 (C0CCTRL)	R				US	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																	
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W				RSV	RSV	RSV	RNG	RSV	MD	RSV	RSV	RSV	RC4	DES	AES	ALL	
0xBASE_803C (C0CIRQ)	R	0	0	0	0	0	0	RNEI	0	MEI	0	0	0	RCEI	DEI	AEI	0	
	W				RSV	RSV	RSV	w1c	RSV	w1c	RSV	RSV	RSV	w1c	w1c	w1c		
	R	0	0	0	0	0	0	RNDI	0	MDI	0	0	0	RCDI	DDI	ADI	0	
	W				RSV	RSV	RSV	w1c	RSV	w1c	RSV	RSV	RSV	w1c	w1c	w1c		
0xBASE_8044 (C0CWR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W										C2K	C2C			C2DS	C2KS		C2M
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	RSV	RSV	RSV	RSV						C1K	C1C		C1CV	C1DS		C1M	
0xBASE_8048 (C0CSTA_MS)	R	CL2				0	0	0	0	0	0	0	0	ERRID2				
	W																	
	R	CL1				0	0	0	0	0	0	0	0	ERRID1				
	W																	
0xBASE_804C (C0CSTA_LS)	R	0	PIZ	GCD	PRM	0	0	0	0	0	0	0	SEI	PEI	0	0	SDI	PDI
	W																	
	R	0	0	0	0	0	0	RNB	CB	MB	PB	SB	KB	RCB	DB	AB	0	
	W																	
0xBASE_805C (C0AADSZR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																	
	R	0	0	0	0	0	0	0	0	0	0	0	0	AASZ				
	W																	
0xBASE_8064 (C0C1IVSZR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																	
	R	0	0	0	0	0	0	0	0	0	0	0	0	IVSZ				
	W																	

## CAAM Register Descriptions

0xBASE_8100 - 0xBASE_817F (C0C1CTXR)	R	C1CTX															
	W																
	R																
	W																
0xBASE_8200 - 0xBASE_827F (C0C1KEYR)	R	C1KEY															
	W																
	R																
	W																
0xBASE_8404 (C0C2MR)	R	0	0	0	0	0	0	0	0	ALG							
	W																
	R	0	0	0	AAI						AS		ICV		AP		
	W																
0xBASE_840C (C0C2KSR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	0	0	0	0	0	0	0	0	C2KS							
	W																
0xBASE_8410 (C0C2DSR_MS)	R	NUMBITS			0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C2CY	
	W																
0xBASE_8414 (C0C2DSR_LS)	R	C2DS															
	W																
	R																
	W																
0xBASE_841C (C0C2ICVSZR)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ICVSZ	
	W																

0xBASE_8500 - 0xBASE_8547 (C0C2CTXR)	R	C2CTX														
	W															
	R															
	W															
0xBASE_8600 - 0xBASE_867F (C0C2KEYR)	R	C2KEY														
	W															
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	DEST	LC2	LC1	FC2	FC1	STYPE 1 10	DTYPE				BND	PTYPE			
0xBASE_87D4 (C0NFIFO)  Format when STYPE ≠ 10	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	OC	AST				DL									
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	OC	AST				BM	PS					PL			
0xBASE_87D4 (C0NFIFO)  Format when STYPE = 10	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	DEST	LC2	LC1	FC2	FC1	STYPE=10	DTYPE				BND	PTYPE			
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	OC	AST				BM	PS					PL			
0xBASE_87E0 (C0IFIFO)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	IFIFO														
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	IFIFO														
0xBASE_87F0 (C0OFIFO_MS)	R	OFIFO_MS														
	W															
	R	OFIFO_MS														
	W															
0xBASE_87F4 (C0OFIFO_LS)	R	OFIFO_LS														
	W															
	R	OFIFO_LS														
	W															

## CAAM Register Descriptions

0xBASE_8800 (D0JQCR_MS)	R	STEP	SING	WHL	FOUR	IIE	SHR_FROM	0	0	0	0	0	0	0	0
	W														
	R	AMTD	JDMS	0	0	0	SRC	0	0	0	0	ID			
	W														
0xBASE_8804 (D0JQCR_LS)	R	CMD													
	W														
	R	CMD													
	W														
0xBASE_880C (D0DAR)	R	DPTR													
	W														
	R	DPTR													
	W														
0xBASE_8810 (D0OPSTA_MS)	R	STATUS_TYPE				NLJ	0	0	0	0	0	0	0	0	0
	W														
	R	0	0	COMMAND_INDEX						STATUS					
	W														
0xBASE_8814 (D0OPSTA_LS)	R	OUT_CT													
	W														
	R	OUT_CT													
	W														
0xBASE_8820 (D0LSR_MS)	R	0	0	0	0	DNSMID									
	W														
	R	0	0	0	0	DSMID									
	W														
0xBASE_8824 (D0LSR_LS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W														
	R	0	0	0	0	DTMID									
	W														
0xBASE_8840 (DOMTH0_MS)	R	MATH_REGISTER_0_MS													
	W														
	R	MATH_REGISTER_0_MS													
	W														
0xBASE_8848 (DOMTH1_MS)	R	MATH_REGISTER_1_MS													
	W														
	R	MATH_REGISTER_1_MS													
	W														

0xBASE_8850 (D0MTH2_MS)	R	MATH_REGISTER_2_MS		
	W			
	R			
	W			
0xBASE_8858 (D0MTH3_MS)	R	MATH_REGISTER_3_MS		
	W			
	R			
	W			
0xBASE_8880 (D0GTR<i>_0), offset 16, range 0 .. 3	R	Reserved (must be 0)		
	W			
	R	Reserved (must be 0)		Reserved (must be 0)
	W			Reserved (must be 0)
0xBASE_8884 (D0GTR<i>_1), offset 16, range 0 .. 3	R	Address Pointer - ms 16 bits		
	W			
	R	Address Pointer - ls 16 bits		
	W			
0xBASE_8888 (D0GTR<i>_2), offset 16, range 0 .. 3	R	E	F	Length - ms 14 bits
	W			
	R	Length - ls 16 bits		
	W			
0xBASE_888C (D0GTR<i>_3), offset 16, range 0 .. 3	R	Reserved (must be 0)		Reserved (must be 0)
	W			
	R	Reserved (must be 0)		Offset
	W			
0xBASE_8900 (D0STR<i>_0), offset 16, range 0 .. 3	R	Reserved (must be 0)		
	W			
	R	Reserved (must be 0)		Reserved (must be 0)
	W			Reserved (must be 0)
0xBASE_8904 (D0STR<i>_1), offset 16, range 0 .. 3	R	Address Pointer - ms 16 bits		
	W			
	R	Address Pointer - ls 16 bits		
	W			

## CAAM Register Descriptions

0xBASE_9088 (D0STR<i>_2), offset 16, range 0 .. 3	R	E	F	Length - ms 14 bits												
	W			Length - ls 16 bits												
	R			Reserved (must be 0)												
	W			Reserved (must be 0)	Offset											
0xBASE_8A00 (D0DESB_<i>), offset 4, range 0 .. 63	R				Descriptor Buffer											
	W				Descriptor Buffer											
	R				Descriptor Buffer											
	W				Descriptor Buffer											
0xBASE_8E00 (D0DJR)	R	STEP	SING	WHL	FOUR	ILE	SHR_FROM		0	0	0	0	0	0	0	0
	W	AMTD	JDMS	0	0	0	SRC	0	0	0	0	ID				
	R											ID				
	W											ID				
0xBASE_8E04 (D0DDR)	R	VALID	SD	TRCT		SEQMSEL	NSEQMSEL	DECO_STATE				PDB_VWB_ST		PDB_STALL		
	W															
	R	PTCL_RUN	NLJ	CURRENT_REAL_ADDR				CMD_STAGE				CSA	NC	BWB	BRB	CT
	W															
0xBASE_8E0C (D0JDP)	R	JDPTR														
	W															
	R	JD PTR														
	W															

0xBASE_8E14 (D0SDP)	R	SDPTR													
	W														
	R	SDPTR													
	W														
0xBASE_8E18 (D0DMR_MS)	R	0	0	0	0	0	0	0	0	0	0	0	DNSMID		
	W														
	R	0	0	0	0	0	0	0	0	0	0	0	DSMID		
	W														
0xBASE_8E1C (D0DMR_LS)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W														
	R	0	0	0	0	0	0	0	0	0	0	0	DTMID		
	W														

## B.2 Register Definitions

The format and fields in each CAAM register are defined below. Some of the register format figures apply to several different registers. In such cases a different parenthesized register name will follow each of the register offset addresses that appear at the top of the register format figure. Although these registers share the same format, they are independent registers. In addition, many registers can be accessed at multiple addresses. In these cases there will be a single parenthesized register name that follows the list of addresses at which that register is accessible. Unless noted in the individual register descriptions, registers are reset only at Power-On Reset (POR).

Although many of the CAAM registers hold more than 32 bits, the diagrams in this document represent how these registers are accessed over the IP bus as 32-bit words. Note that all registers other than the Input Data FIFO must be accessed only as full 32-bit words. Byte enables are permitted only for the Input Data FIFO, and this data must be left-aligned. Only register words containing currently defined fields are represented in the register diagrams (that is, words containing only reserved fields have been omitted from these diagrams). The addresses shown below correspond to the lowest-address portion of the register that holds currently defined fields. All addresses not shown are reserved.

Data read from and written to DECO registers by software is treated as control data for the purpose of endianess conversion.

## B.2.1 Configuration and Special Key Registers

This section describes general configuration, status and control registers of the CAAM, and special key registers that are accessible only when CAAM is in Non-secure Mode. These registers are intended to be accessed only at boot time or during debugging.

### B.2.1.1 Master Configuration Register (MCFGR)

The Master Configuration Register is used to set some global CAAM configurations. This register is typically written at boot time, and in some debug scenarios.

Offset 0xBASE_0004 (MCFGR)																Access: Read/Write Accessible only by the Manager Processor			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
W	SWRST	WDE	WDF	0	WRHD	0	0	0	0	0	0	0	0	0	0	0			
PO Reset	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	ARCACHE				AWCACHE				AXIPIPE				0	LARGE_BURST	0				
W														0		0			
PO Reset	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Figure B-3. Master Configuration Register - Format

Table B-14. Master Configuration Register - Field Descriptions

Field	Description
31 SWRST	Software Reset. Writing a 1 to this bit will cause most registers and state machines in CAAM to reset. Note that SWRST will remain 1 (and the registers will be held in reset) until any outstanding CAAM DMA transactions complete. Writing a 1 to SWRST will not cause a reset of the CAAM DMA unless SWRST is already 1 and a 1 is also written to DMARST. Note that writing to MCFGR will overwrite the values in BURST, AXIPIPE, AWCACHE and ARCACHE, so to avoid disrupting outstanding DMA transactions when initiating a SWRST, these fields should be written with their current values.
30 WDE	DECO Watchdog Enable. Enables the DECO Watchdog Timer to run. The Timer is used to detect and flush a job that has caused a DECO to hang. If the DECO Watchdog Timer expires, the hung job is usually flushed from the DECO with an error status indication. In those cases in which a hung job is not flushed automatically, software can reset the DECO via the DECO Reset Register. Note that the watchdog expiration period is extended for RNG reseeding, because these can take longer than the normal watchdog expiration period.
29 WDF	Causes the DECO Watchdog Timer to expire prematurely for testing purposes.

**Table B-14. Master Configuration Register - Field Descriptions**

28 DMA RST	DMA Reset. If SWRST is already 1, writing a 1 to DMARST and SWRST on the same cycle will cause the CAAM DMA to be reset. The DMA will not be reset if SWRST is not already a 1 (i.e. a 1 was previously written to SWRST but DMA transactions have not completed). Following a DMA reset, system software should delay long enough for outstanding AXI transaction responses to finish. These orphaned responses will be ignored.
27 WRHD	Write Handoff Disable. If WRHD=0, when DECO has initiated the last write transaction of the current job DECO will go idle without waiting for the bus slave's response to that write transaction. This allows DECO to start another job while awaiting the slave's response. If an error response is eventually received CAAM will update the transaction status appropriately. If WRHD=1 DECO will wait for the bus slave's response to the last write transaction before DECO goes to the idle state. Setting WRHD=1 is intended for product testing, so WRHD should normally be left at its PO reset value.
26-17	Reserved.
16	Reserved.
15-12 AR CACHE	AXI Read Cache Control. This field sets the 4-bit value for the ARCACHE[3:0] interface signals for read transactions. These signals control the cacheable and allocate attributes of read transactions. For more information refer to the AMBA AXI Protocol Specification. If read-safe is enabled for a read transaction, CAAM will drive ARCACHE[1] to 1, independent of the value of bit 13.  Bit 15 is always 0, regardless of what is written. The four patterns x10xb are reserved.
11-8 AW CACHE	AXI Write Cache Control. This field sets the 4-bit value for the AWCACHE[3:0] interface signals for write transactions. These signals control the bufferable, cacheable, and allocate attributes of writes transaction. For more information refer to the AMBA AXI Protocol Specification. CAAM issues non-bufferable writes when it needs to verify that memory has been written. For those writes, the AXI AWCACHE[0] signal will be forced to 0, independent of the value of bit 8.  Bit 10 is always 0, regardless of what is written. The two patterns 1000b and 1001b are reserved.
7-4 AXPIPE	AXI Pipeline Depth. This field limits the number of outstanding transactions on the AXI interface. These are read and/or write transactions that have been started but not completed (response not yet received). The capability to issue multiple AXI transactions is especially useful for reads since there may be a significant delay before completion of read transactions. Pipelining effectively allows transactions to overlap so that the delay between read and write responses is reduced. Setting this field to 0 is equivalent to setting it to a value of 1.
3	Reserved.
2 LARGE_ BURST	Enable Large Bursts. When LARGE_BURST=1, Shared Descriptor reads can use transactions as large as the maximum AXI interface transaction size, which equals 16 times the width of the AXI data buses (i.e. 64 bytes for 32-bit data buses). When LARGE_BURST=0, all master bus transactions use the normal burst size (which is visible in the NORMAL_BURST field). Changes to LARGE_BURST should be made only when CAAM is not processing jobs.
1	Reserved
0	Reserved

### B.2.1.2 Security Configuration Register (SCFGR)

The Security Configuration Register is used to set security-related mode bits. These bits are used to switch from special boot-time operating modes to normal operating modes. At POR, all bits in SCFGR reset to 0.

When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in deterministic mode. This allows the High Assurance Boot software to run deterministic tests on the RNG and its State Handle 0 logic. Once the tests have been completed, the High Assurance Boot software can write a 1 to RNGSH0 to prevent State Handle 0 from being instantiated in deterministic mode. This ensures that random data, rather than deterministic data, is used for the Zeroizable Master Key Register in the low-power section of SNVS, the Differential Power Analysis Resistance Mask in the AESA, the Job Descriptor Key Encryption Register, the Trusted Descriptor Key Encryption Register, the Trusted Descriptor Signing Key Register and the random padding used by built-in protocols.

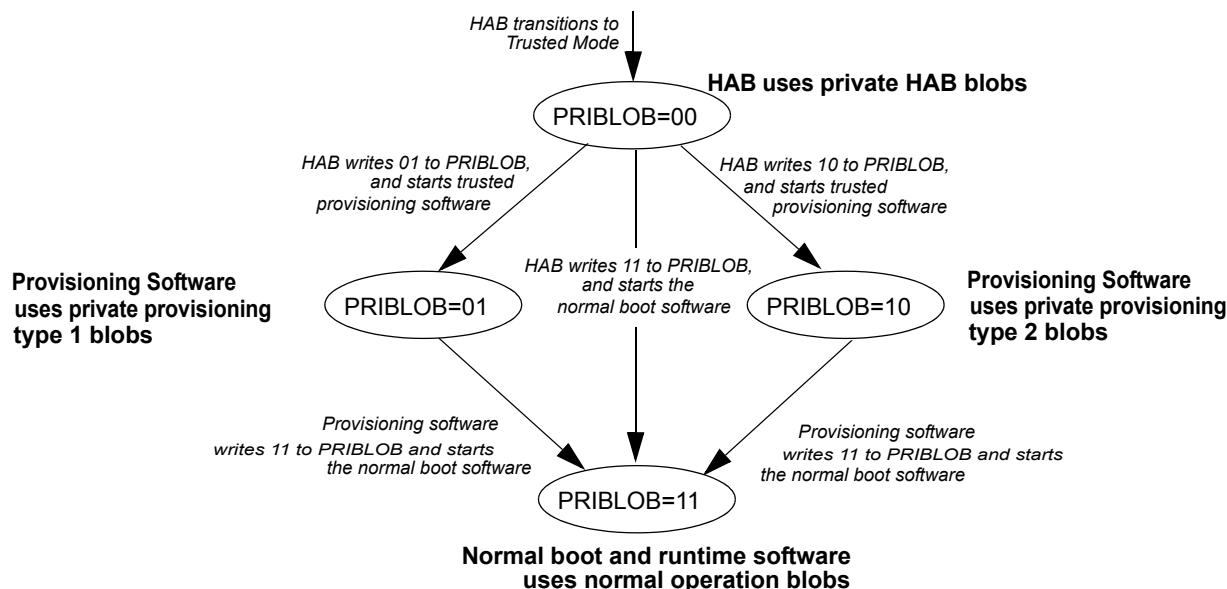
At POR the AESA will seed its Differential Power Analysis Resistance Mask using a default deterministic value. Writing a 1 to the RANDDPAR bit will cause the AESA to reseed its Differential Power Analysis Resistance Mask using data from RNG DRNG State Handle 0. Typically the High Assurance Boot software will write a 1 to RANDDPAR after RNG DRNG State Handle 0 has been instantiated in a non-deterministic mode. Once a 1 has been written to RANDDPAR, this bit will remain 1 until the next POR. Software can read this bit to determine whether the Differential Power Analysis Resistance Mask was seeded from the RNG.

The PRIBLOB field is used to select a private blob type during Trusted Mode. When a General Memory Blob or Secure Memory Blob is encapsulated or decapsulated during Trusted Mode, the PRIBLOB bits are used to modify the derivation of the Blob Key Encryption Key (see Blob encapsulation). This is used to enforce cryptographic separation of private blob types during the boot process (and thereafter). These bits reset to 0 at POR, but when a PRIBLOB bit is written to a 1, it remains a 1 until the next POR.

The PRIBLOB=00 setting allows High Assurance Boot Software to have its own private blobs that cannot be decapsulated or encapsulated by other software, even software that later runs in Trusted Mode. This feature can be used to safeguard boot reference metrics (e.g. hash values over software). In this use case, the reference metrics might be initially verified via a public key signature and then encapsulated in a private HAB blob. On subsequent boot cycles the protected reference metrics would be obtained by decapsulating the private HAB blob, obviating the time-consuming public key signature verification process.

The PRIBLOB=01 and PRIBLOB=10 settings allow trusted provisioning software (e.g. software that handles DRM keys) to have private blobs that cannot be decapsulated or encapsulated by software that runs later in the boot process, even if that software runs in Trusted Mode.

As illustrated in [Figure B-4](#), typically the HAB software would enter Trusted Mode, then encapsulate or decapsulate all of its private blobs, and then would write either a 01, 10 or 11 to PRIBLOB. For the remainder of the current power-on session, private HAB blobs could no longer be encapsulated or decapsulated. HAB would then either run provisioning software with the 10 or 01 setting, or would skip the provisioning software and run the normal boot software with the 11 setting. If the provisioning software runs, it would encapsulate or decapsulate its own private blobs and then write 11 to PRIBLOB. At this point PRIBLOB=11 for the remainder of the current power on session, and no software can encapsulate or decapsulate private HAB blobs or either type of private provisioning blobs.



**Figure B-4. Process for Managing Private Blobs**

## CAAM Register Descriptions

Offset 0xBASE\_000C (SCFGR)

Access: Read/Write  
Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	RDB	RNGSH0	RANDDPAR	0	0	0	0	0	0	0	PRIBLOB
W					w1s	w1s	w1s							w1s	w1s	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-5. Security Configuration Register - Format**

**Table B-15. Security Configuration Register - Field Descriptions**

Field	Description
31-16	Reserved
15-11	Reserved
10	Reserved
9 RNGSH0	Random Number Generator State Handle 0. 0: RNG DRNG State Handle 0 can be instantiated in any mode. 1: RNG DRNG State Handle 0 cannot be instantiated in deterministic (test) mode. If it is currently instantiated in a deterministic mode, it will be un-instantiated. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.
8 RANDDPAR	Random Differential Power Analysis Resistance (DPAR) Mask. 0: The AESA DPAR Mask was seeded using the default deterministic seed 1: When RANDDPAR is written with a 1, the AESA DPAR Mask is reseeded from RNG DRNG State Handle 0. Once RANDDPAR has been written with a 1, it cannot be changed to a 0 until the next power on reset.
7-2	Reserved.
1-0 PRIBLOB	Private Blob. This field selects one of four different types of private blobs during Trusted Mode. All blobs encapsulated or decapsulated during Trusted Mode will be of the type specified in this field, until a 1 is written to any or the bits, or until the next POR. The bits of this field are “sticky”, i.e. once a bit has been written to a 1, it cannot be changed to a 0 until the next power on reset. 00 - private HAB blobs 01 - private provisioning type 1 blobs 10 - private provisioning type 2 blobs 11 - normal operation blobs

### B.2.1.3 Job Ring MID Register (JRMIDR)

There is one JRMIDR register per Job Ring. The Job Ring MID Register is used to specify the MID values that the CAAM DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular Job Ring. This register also contains a bit that grants permission for Trusted Descriptors to be created in this Job Ring. This register is typically written at boot time and then locked.

Note that there is one copy of this register per Job Ring.

Offset 0xBASE\_0010 (JR0MIDR\_MS)(Used with Job Ring 0)  
0xBASE\_0018 (JR1MIDR\_MS)(Used with Job Ring 1)

Access: Read/Write  
Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AMTD
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JROWN
W																
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JROWN
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-6. Job Ring MID Register\_MS - Format**

Note that there is one copy of this register per Job Ring.

Offset 0xBASE\_0014 (JR0MIDR\_LS)(Used with Job Ring 0)  
0xBASE\_001C (JR1MIDR\_LS)(Used with Job Ring 1)

Access: Read/Write  
Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JRNSMID
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	JRSMID
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-7. Job Ring MID Register\_LS - Format**

**Table B-16. Job Ring MID Register - Field Descriptions**

Field	Description
Most-significant half of Job Ring MID Register	
31 LCK	Lock. Once LCK has been set, no further changes can be made to JRxMIDR_MS (including changes to LCK) or to JRxMIDR_LS.
30-17	Reserved.
16 AMTD	Allow Make Trusted Descriptor. If AMTD is set, the Job Ring associated with this register is permitted to issue jobs that create Trusted Descriptors. When DECO encounters a descriptor header with the MTD bit set, the commands in the descriptor will not be executed but DECO will generate a signature and append it following the SIGNATURE command at the end of the descriptor. If AMTD is not set, then executing a descriptor with the MTD bit set in the descriptor's header will result in an error, and no signature will be generated.
4-0 JROWN	Job Ring Owner This field defines the MID of the bus master that is permitted to read or write the registers that are specific to a particular Job Ring. These registers include the Job Ring configuration registers, the interrupt registers, the CAAM Secure Memory Access Permissions and Secure Memory Access Groups registers and the ring buffer registers. The most-significant bit of this field is always 0 in this version of CAAM.
15-4	Reserved.
3-0 JROWN	Job Ring Owner. This field defines the MID of the bus master that is permitted to read or write the registers that are specific to a particular Job Ring. These registers include the Job Ring configuration registers, the interrupt registers, the CAAM Secure Memory Access Permissions and Secure Memory Access Groups registers and the ring buffer registers.
Least-significant half of Job Ring MID Register	
31-20	Reserved.
27-16 JRNSMID	Job Ring Non-SEQ MID This field defines the MID value asserted for DMA transactions associated with external memory accesses for non-sequence commands, like KEY, LOAD, and STORE. By default the Job Descriptor is read using this LIODN, although that behavior can be changed by setting the JDLS bit in the corresponding Job Ring Configuration Register.
15-12	Reserved.
3-0 JRSMID	Job Ring SEQ MID This field defines the MID value asserted for DMA transactions associated with external memory accesses for sequence commands, like SEQ_KEY, SEQ_LOAD, and SEQ_STORE. Setting the JDLS bit in the corresponding Job Ring Configuration Register will cause the Job Queue to use this MID for Job Descriptor reads.
31-20	Reserved.
19-16 JRNSMID	Job Ring Non-SEQ MID This field defines the MID value asserted for DMA transactions associated with external memory accesses for non-sequence commands, like KEY, LOAD, and STORE. By default the Job Descriptor is read using this LIODN, although that behavior can be changed by setting the JDLS bit in the corresponding Job Ring Configuration Register.

**Table B-16. Job Ring MID Register - Field Descriptions**

Field	Description
15-4	Reserved.
3-0 JRSMD	Job Ring SEQ MID This field defines the MID value asserted for DMA transactions associated with external memory accesses for sequence commands, like SEQ_KEY, SEQ_LOAD, and SEQ_STORE. Setting the JDLS bit in the corresponding Job Ring Configuration Register will cause the Job Queue to use this MID for Job Descriptor reads.

### B.2.1.4 DECO Request Register (DECORR)

This register is used when software wants to bypass the normal Job Queue Controller mechanism and directly access the DECO/CCB block. Since this interface is not as efficient as the Job Ring interface, this interface would normally be used only for debugging and testing purposes. The register and its fields are described in [Figure B-8](#) and [Table B-17](#).

Offset 0xBASE_009C (DECORR)																Access: Read/Write Accessible only by the Manager Processor				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DEN 0				
Reset <sub>1</sub>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RQD 0				
Reset <sub>1</sub>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-8. DECO Request Register - Format**

<sup>1</sup> Reset occurs at POR & SWRST

**Table B-17. DECO Request Register - Field Descriptions**

Field	Description
31-17	Reserved. Always 0.
16 DEN0	The Job Queue Controller asserts this bit when permission is granted for the software to directly access DECO 0/CCB 0.
15-1	Reserved. Always 0.
0 RQD0	This bit is set by software to request direct access to DECO 0/CCB 0. It must not be cleared until the direct access operation is complete.

### B.2.1.5 DECO MID Register (DECOxMID)

The DECO MID register is used to specify the MID values that the CAAM DMA asserts when reading or writing memory on behalf of a DECO that is under the direct control of software. This register is also used to specify the MID that must be asserted by the processor in order to read or write the registers that are specific to the DECO. This register is intended to be written by the same processor that writes to the DECORR.

Offset 0xBASE\_00A0 (DECO0MID\_MS)

 Access: Read/Write  
 Accessible only by the Manager Processor

R				31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W				LCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W				0	0	0	0	0	0	0	0	0	0	0	0	DOWN			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-9. DECO MID Register\_MS - Format**

Offset 0xBASE\_00A4 (DECO0MID\_LS)

Access: Read/Write  
Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	DNSMID											
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	DSMID											
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-10. DECO MID Register\_LS - Format

Table B-18. DECO MID Register - Field Descriptions

Field	Description
Most-significant half of DECO MID Register	
31 LCK	Lock. Once LCK is set, no further changes can be made to DECOxMIDR (including any changes to LCK).
30-5	Reserved.
4-0 DOWN	DECO Owner. This defines the MID that is allowed to read or write the registers that are specific to a particular DECO. The most-significant 8 bits of this field are always 0 in this version of CAAM.
Least-significant half of DECO MID Register	
31-28	Reserved.
27-16 DNSMID	DECO Non-SEQ MID. This defines the MID value that CAAM DMA will assert on the AXI Master interface when this DECO is under the direct control of software and the DECO is performing reads and writes during operations other than SEQ operations. 8
15-12	Reserved.
11-0 DSMID	DECO SEQ MID. This defines the MID value that CAAM DMA will assert on the AXI Master interface when this DECO is under the direct control of software and the DECO is performing reads and writes during SEQ operations. 8

### B.2.1.6 DECO Availability Register (DAR)

The DECO Availability Register can be used to determine whether the DECO is hung. If software writes a 1 to the DECO's NYA field, the DECO will clear that bit whenever the

DECO is, or becomes, available. The bit can be polled to determine if the DECO is completing jobs.

Offset 0xBASE_0120 (DAR)																Access: Read/Write Accessible only by the Manager Processor				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	
W																			NYAO	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-11. DAR Register - Format

Table B-19. DAR Register - Field Descriptions

Field	Description
31-1	Reserved. Always 0.
0 NYAO	This bit is set by software to start polling for the availability of DECO 0. This bit will be reset when DECO 0 is or becomes, available.

### B.2.1.7 DECO Reset Register (DRR)

The DECO Reset Register can be used to force a soft reset of the DECO with appropriate status write back (error code 20h). Note that using this can result in lost DMA transactions and/or memory leaks. In some cases a soft reset of a DECO will not result in a status write back, or may not free a hung DECO. If a hung DECO cannot be freed via a soft DECO reset, then a software CAAM reset or a POR will be required.

Offset 0xBASE_0124 (DRR)																Access: Write-Only Accessible only by the Manager Processor				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	
W																			RST0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-12. DRR Register - Format

**Table B-20. DRR Register - Field Descriptions**

Field	Description
31-1	Reserved. Always 0.
0 RST0	Software writes a 1 to this bit to initiate a soft reset of DECO 0. This bit is self-clearing after one clock cycle.

### B.2.1.8 Job Descriptor Key Encryption Key Register (JDKEKR)

The Job Descriptor Key Encryption Key Register contains the Job Descriptor Key Encryption Key (JDKEK), which can be used when encrypting or decrypting Black Keys (see ). Since Black Keys are not intended for storage of keys across chip power cycles (CAAM’s Blob mechanism is intended for this purpose), the value in the JDKEKR is not preserved at SOC power-down. Instead, a new 256-bit secret value is loaded into the JDKEKR from the RNG at power-on for use during the current power-on session. The JDKEK is loaded by executing a special descriptor. The JDKEKR cannot be read or written while CAAM is in Secure Mode or Trusted Mode, but JDKEKR can be read and written while CAAM is in Nonsecure Mode. Note that the Secure Mode/Trusted Mode value in JDKEKR is not available when CAAM is in Nonsecure Mode because the only possible transitions between Trusted Mode or Secure Mode that lead to Nonsecure Mode cause CAAM to pass through Fail Mode, and JDKEKR is cleared whenever CAAM enters Fail Mode.

The Job Descriptor Key Encryption Key is 256 bits, so it requires eight 32-bit registers to implement. Offset 0400 contains bits 255-224, offset 0404 contains bits 223-192, offset 0408 contains bits 191-160, Offset 040C contains bits 159-128, offset 0410 contains bits 127-96, offset 0414 contains bits 95-64, offset 0418 contains bits 63-32, and offset 041C contains bits 31-0.

Offset 0xBASE\_0400 (JDKEKR\_<i>), offset 4, range 0 .. 7

Access: Read/Write  
(in Non-Secure Mode)  
Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JDKEK															
W																
PO Reset <sup>1</sup>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

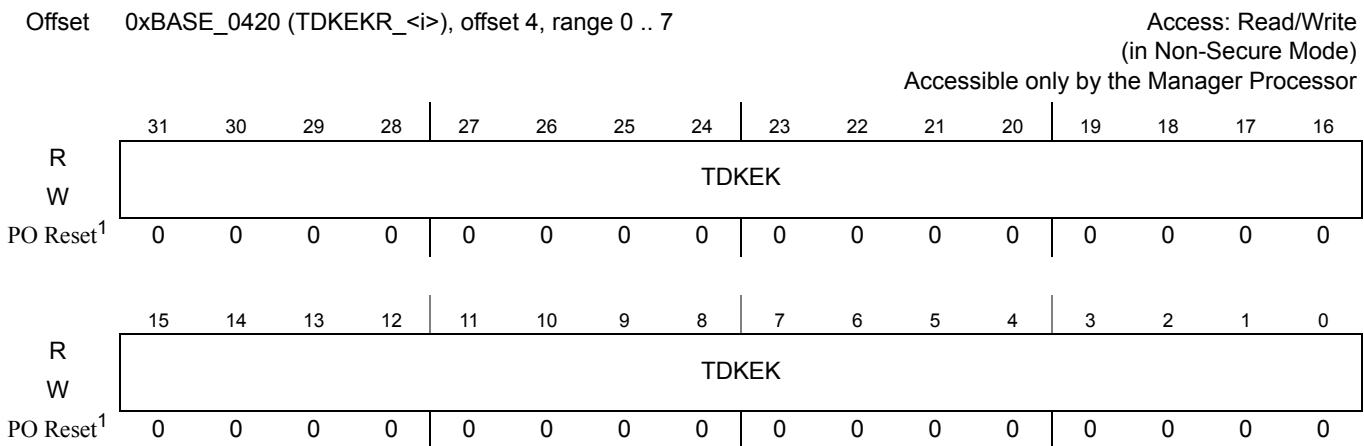
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JDKEK															
W																
PO Reset <sup>1</sup>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-13. JDKEK Register - Format**

<sup>1</sup> Register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

### B.2.1.9 Trusted Descriptor Key Encryption Key Register (TDKEKR)

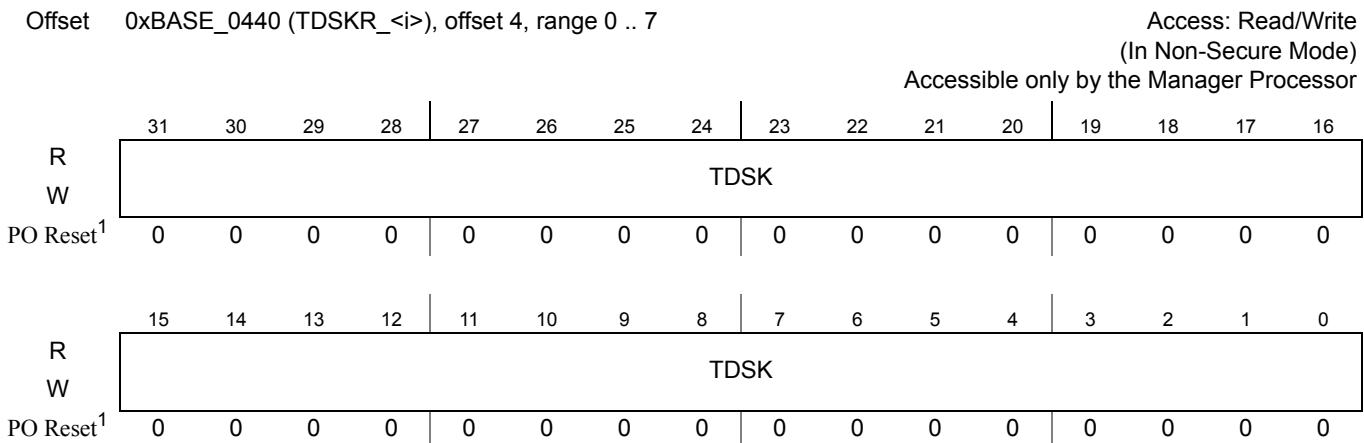
The Trusted Descriptor Key Encryption Key Register contains the Trusted Descriptor Key Encryption Key (TDKEK), which can be used when encrypting or decrypting Black Keys (see Black keys). The TDKEKR operates exactly like the JDKEKR, except that the TDKEKR is usable only by Trusted Descriptors. This allows Trusted Descriptors to protect particularly sensitive keys from access by Job Descriptors. Trusted Descriptors can use either the JDKEKR or the TDKEKR, so Trusted Descriptors can be used to derive non-Trusted Black keys for use by Job Descriptors from Trusted Black Keys that contain master secrets. The Trusted Descriptor Key Encryption Key is 256 bits, so it requires eight 32-bit registers to implement. The bit numbering is the same as for the JDKEKR.

**Figure B-14. TDKEK Register - Format**

<sup>1</sup> Register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

### B.2.1.10 Trusted Descriptor Signing Key Register (TDSKR)

The Trusted Descriptor Signing Key Register contains the TDSK, which is used to generate and verify signatures on Trusted Descriptors. The TDSK is 256 bits, so it requires eight 32-bit registers to implement. The bit numbering is the same as for the JDKEKR.

**Figure B-15. TDSK Register - Format**

<sup>1</sup> Register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

### B.2.1.11 Secure KeyNonce Register (SKNR)

The Secure KeyNonce Register holds a nonce value that is used for Black Key encryption. The nonce is used and incremented whenever a Black Key is encrypted using AES-CCM encryption (i.e. a FIFO STORE with EKT=1, of the AFHA S-Box, the Class 1 Key Register or the Class 2 Key Register.) The register is reset to all 0 at power on reset or when CAAM enters Fail mode, but it is not reset at software-initiated CAAM reset. Since the SKNR holds 45 bits, it is accessed over the IP bus as two 32-bit words. The DECO identification number is appended as the three least-significant bits to form a 48-bit nonce value that is used in the AES-CCM algorithm.

Offset 0xBASE_04E0 (SKNR_MS)																Access: Read/Write (in Non-Secure Mode)				
																Accessible only by the Manager Processor				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
W																	SK_NONCE_MS			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-16. Secure KeyNonce Register\_MS - Format

Offset 0xBASE_04E4 (SKNR_LS)																Access: Read/Write (in Non-Secure Mode)				
																Accessible only by the Manager Processor				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
W																	SK_NONCE_LS			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
W																	SK_NONCE_LS			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-17. Secure KeyNonce Register\_LS - Format

**Table B-21. Secure Key Nonce Register - Field Descriptions**

Field	Description
Most-significant portion of Secure Key Nonce Register	
31-13	Reserved.
12-0 SK_NONCE_MS	Secure KeyNonce - Most Significant Bits. This field holds the 13 most-significant bits of the SKNR.
Least-significant portion of Secure Key Nonce Register	
31-0 SK_NONCE_LS	Secure KeyNonce - Least Significant Bits. This field holds the 32 least-significant bits of the SKNR. The nonce value that is used during AES-CCM encryption of Black Keys is the 48-bit value $SK\_NONCE\_MS_{13} \parallel SK\_NONCE\_LS_{32} \parallel DECO\_ID_3$ .

## B.2.2 RNG Registers

These registers are intended to be used when testing the RNG. They would not be used during normal operation. During normal operation the RNG is configured and data is obtained from the RNG via Job Descriptors.

### B.2.2.1 RNG TRNG Miscellaneous Control Register (RTMCTL)

The RNG TRNG Miscellaneous Control Register is a read/write register used to control the RNG's True Random Number Generator (TRNG) access, operation and test. *Note that in many cases two RNG registers share the same address, and a particular register at the shared address is selected based upon the value in the PRGM field of the RTMCTL register.*

## CAAM Register Descriptions

Offset 0xBASE\_0600 (RTMCTL)

Access: Read/Write

Accessible only by the Manager Processor

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PRGM
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W			TSTOP_OK	ERR	TST_OUT	ENT_VAL	FCT_VAL	FCT_FAIL	FORCE_SYSCLK	0	TRNG_ACC	CLK_OUT_EN	OSC_DIV	SAMP_MODE		
PO Reset	0	0	0	0	w1c	0	0	0	0	0	0	0	0	0	0	0

Figure B-18. RNG TRNG Miscellaneous Control Register - Format

Table B-22. RNG TRNG Miscellaneous Control Register - Field Descriptions

Field	Description
31-17	Reserved.
16 PRGM	Programming Mode Select. When this bit is 1, the TRNG is in Program Mode, otherwise it is in Run Mode. No Entropy value will be generated while the TRNG is in Program Mode. <b>Note that different RNG registers are accessible at the same address depending on whether PRGM is set to 1 or 0.</b> This is noted in the RNG register descriptions.
15-14	Reserved.
13 TSTOP_OK	TRNG_OK_TO_STOP. This bit is asserted when the TRNG ring oscillator (used for entropy generation) is not running, and therefore it is ok to do ipg_stop.
12 ERR	Read: Error status. 1 = error detected. 0 = no error. Write: Write 1 to clear errors. Writing 0 has no effect.
11 TST_OUT	Read only: Test point inside ring oscillator.
10 ENT_VAL	Read only: Entropy Valid. Will assert only if TRNG ACC bit is set, and then after an entropy value is generated. Will be cleared when RTENT11 is read. (RTENT0 through RTENT10 should be read before reading RTENT11).
9 FCT_VAL	Read only: Frequency Count Valid. Indicates that a valid frequency count may be read from RTFRQCNT.
8 FCT_FAIL	Read only: Frequency Count Fail. The frequency counter has detected a failure. This may be due to improper programming of the RTFRQMAX and/or RTFRQMIN registers, or a hardware failure in the ring oscillator. This error may be cleared by writing a 1 to the ERR bit.
7 FORCE_SYSCLK	Force System Clock. If set, the system clock is used to operate the TRNG, instead of the ring oscillator. This is for test use only, and indeterminate results may occur. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. This bit is cleared by writing the RST_DEF bit to 1.
6 RST_DEF	Reset Defaults. Writing a 1 to this bit clears various TRNG registers, and bits within registers, to their default state. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. Reading this bit always produces a 0.

Field	Description
5 TRNG_ACC	TRNG Access Mode. If this bit is set to 1, the TRNG will generate an Entropy value that can be read via the RTENT0-RTENT11 registers. The Entropy value may be read once the ENT VAL bit is asserted. This Entropy value will never be used by the RNG.  IMPORTANT: If this bit is set, no Entropy value can be generated for the RNG, which can prevent the RNG from generating data for the CAAM system.
4 CLK_OUT_EN	Clock Output Enable. If set, the ring oscillator output is gated to an output pad. If this bit is set and PRGM mode is selected, this allows external viewing of the ring oscillator.
3-2 OSC_DIV	Oscillator Divide. Determines the amount of dividing done to the ring oscillator before it is used by the TRNG.  00:use ring oscillator with no divide 01:use ring oscillator divided-by-2 10:use ring oscillator divided-by-4 11:use ring oscillator divided-by-8  This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this field. This field is cleared to 00 by writing the RST_DEF bit to 1.
1-0 SAMP_MODE	Sample Mode. Determines the method of sampling the ring oscillator while generating the Entropy value:  00:use Von Neumann data into both Entropy shifter and Statistical Checker 01:use raw data into both Entropy shifter and Statistical Checker 10:use Von Neumann data into Entropy shifter. Use raw data into Statistical Checker 11:undefined/reserved.  This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously with writing this field. This field is cleared to 00 by writing the RST_DEF bit to 1.

### B.2.2.2 RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)

The RNG TRNG Statistical Check Miscellaneous Register contains the Long Run Maximum Limit value and the Retry Count value. This register is accessible only when the RTMCTL[PRGM] bit is 1, otherwise this register will read zeroes, and cannot be written.

Offset	0xBASE_0604 (RTSCMISC)	Access: Read/Write
Accessible only by the Manager Processor		
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	RTY_CNT
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 1	
Reset <sup>1</sup>	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 1	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	LRUN_MAX
W	0 0 0 0   0 0 0 0   0 0 1 0   0 0 1 0	
Reset <sup>1</sup>	0 0 0 0   0 0 0 0   0 0 1 0   0 0 1 0	

Figure B-19. RNG TRNG Statistical Check Miscellaneous Register - Format

<sup>1</sup> Reset occurs at POR, and when RTMCTL[RST\_DEF] is written to 1.

**Table B-23. RNG TRNG Statistical Check Miscellaneous Register - Field Descriptions**

Field	Description
31-20	Reserved.
19-16 RTY_CNT	RETRY COUNT. If a statistical check fails during the TRNG Entropy Generation, the RTY_CNT value indicates the number of times a retry should occur before generating an error. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 1h by writing the RTMCTL[RST_DEF] bit to 1.
15-8	Reserved.
7-0 LRUN_MAX	LONG RUN MAX LIMIT. This value is the largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 22h by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.3 RNG TRNG Poker Range Register (RTPKRRNG)

The RNG TRNG Poker Range Register defines the difference between the TRNG Poker Maximum Limit and the minimum limit. These limits are used during the TRNG Statistical Check Poker Test.

Offset	0xBASE_0608 (RTPKRRNG)	Access: Read/Write Accessible only by the Manager Processor	
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16		
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0		
R	PKR_RNG		
W			
PO Reset	0 0 0 0   1 0 0 1   1 0 1 0   0 0 1 1		

**Figure B-20. RNG TRNG Poker Range Register - Format****Table B-24. RNG TRNG Poker Range Register - Field Descriptions**

Field	Description
31-16	Reserved. Always 0.
15-0 PKR_RNG	Poker Range. During the TRNG Statistical Checks, a “Poker Test” is run which requires a maximum and minimum limit. The maximum is programmed in the RTPKRMAX[PKR_MAX] register, and the minimum is derived by subtracting the PKR_RNG value from the programmed maximum value. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09A3h (decimal 2467) by writing the RTMCTL[RST_DEF] bit to 1. Note that the minimum allowable Poker result is PKR_MAX - PKR_RNG + 1.

### B.2.2.4 RNG TRNG Poker Maximum Limit Register (RTPKRMAX)

The RNG TRNG Poker Maximum Limit Register defines Maximum Limit allowable during the TRNG Statistical Check Poker Test. Note that this address ( $_{0x}$ BASE\_060C) is used as RTPKRMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTPKRSQ readback register, as described in the next section.

Offset	0x $_{0x}$ BASE_060C (RTPKRMAX) (Accessible at this address when RTMCTL[PRGM] = 1)																Access: Read/Write
																	Accessible only by the Manager Processor
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	0	0	0	0	0	0	0	0	PKR_MAX								
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	PKR_MAX																
PO Reset	0	1	1	0	1	0	0	1	0	0	1	0	0	0	0	0	

**Figure B-21. RNG TRNG Poker Maximum Limit Register - Format**

**Table B-25. RNG TRNG Poker Maximum Limit Register - Field Descriptions**

Field	Description
31-24	Reserved. Always 0.
23-0 PKR_MAX	Poker Maximum Limit. During the TRNG Statistical Checks, a “Poker Test” is run which requires a maximum and minimum limit. The maximum allowable result is programmed in the RTPKRMAX[PKR_MAX] register. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 006920h (decimal 26912) by writing the RTMCTL[RST_DEF] bit to 1. Note that the RTPKRMAX and RTPKRRNG registers combined are used to define the minimum allowable Poker result, which is PKR_MAX - PKR_RNG + 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Poker Test Square Calculation result in register RTPKRSQ, as defined in the following section.

### B.2.2.5 RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)

The RNG TRNG Poker Square Calculation Result Register is a read-only register used to read the result of the TRNG Statistical Check Poker Test’s Square Calculation. This test starts with the RTPKRMAX value and decreases towards a final result, which is read here. For the Poker Test to pass, this final result must be less than the programmed RTPKRRNG value. Note that this address ( $_{0x}$ BASE\_060C) is used as RTPKRMAX if RTMCTL[PRGM] is

1. If RTMCTL[PRGM] is 0, this address is used as RTPKRSQ readback register, as described here.

Offset	0xBASE_060C (RTPKRSQ)																Access: Read-Only Accessible only by the Manager Processor
	(Accessible at this address when RTMCTL[PRGM] = 0)																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	PKR_SQ
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PKR_SQ
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-22. RNG TRNG Poker Square Calculation Result Register - Format**

**Table B-26. RNG TRNG Poker Square Calculation Result Register - Field Descriptions**

Field	Description
31-24	Reserved. Always 0.
23-0 PKR_SQ	Poker Square Calculation Result. During the TRNG Statistical Checks, a “Poker Test” is run which starts with the value RTPKRMAX[PKR_MAX]. This value decreases according to a “sum of squares” algorithm, and must remain greater than zero, but less than the RTPKRRNG[PKR_RNG] limit. The resulting value may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Poker Test Maximum Limit in register RTPKRMAX, as defined in the previous section.

### B.2.2.6 RNG TRNG Seed Control Register (RTSDCTL)

The RNG TRNG Seed Control Register contains two fields. One field defines the length (in system clocks) of each Entropy sample (ENT\_DLY), and the other field indicates the number of samples that will taken during each TRNG Entropy generation (SAMP\_SIZE).

Offset	0xBASE_0610 (RTSDCTL)																Access: Read/Write Accessible only by the Manager Processor
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	ENT_DLY
W	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	
PO Reset	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	SAMP_SIZE
W	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	
PO Reset	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	

**Figure B-23. RNG TRNG Seed Control Register - Format**

**Table B-27. RNG TRNG Seed Control Register - Field Descriptions**

Field	Description
31-16 ENT_DLY	Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 00C8h (decimal 200) by writing the RTMCTL[RST_DEF] bit to 1.
16-0 SAMP_SIZE	Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09C4h (decimal 2500) by writing the RTMCTL[RST_DEF] bit to 1.

**B.2.2.7 RNG TRNG Sparse Bit Limit Register (RTSBLIM)**

The RNG TRNG Sparse Bit Limit Register is used when Von Neumann sampling is selected during Entropy Generation. It defines the maximum number of consecutive Von Neumann samples which may be discarded before an error is generated. Note that this address (0xBASE\_0614) is used as RTSBLIM only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTTOTSAM readback register, as described in the next section.

Offset	0xBASE_0614 (RTSBLIM)	Access: Read/Write (Accessible at this address when RTMCTL[PRGM] = 1)	Access: Read/Write (Accessible only by the Manager Processor)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16		
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	SB_LIM	
W	0 0 0 0   0 0 0 0   0 0 1 1   1 1 1 1	SB_LIM	
PO Reset	0 0 0 0   0 0 0 0   0 0 1 1   1 1 1 1	SB_LIM	

**Figure B-24. RNG TRNG Sparse Bit Limit Register - Format****Table B-28. RNG TRNG Sparse Bit Limit Register - Field Descriptions**

Field	Description
31-10	Reserved. Always 0.
9-0 SB_LIM	Sparse Bit Limit. During Von Neumann sampling (if enabled by RTMCTL[SAMP_MODE]), samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy. The Sparse Bit Limit defines the maximum number of consecutive samples that may be discarded before an error is generated. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 03hF by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Total Samples count in register RTTOTSAM, as defined in the following section.

### B.2.2.8 RNG TRNG Total Samples Register (RTTOTSAM)

The RNG TRNG Total Samples Register is a read-only register used to read the total number of samples taken during Entropy generation. It is used to give an indication of how often a sample is actually used during Von Neumann sampling. Note that this address (0xBASE\_0614) is used as RTSBLIM if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTTOTSAM readback register, as described here.

Offset	0xBASE_0614 (RTTOTSAM) (Accessible at this address when RTMCTL[PRGM] = 0)																Access: Read-Only Accessible only by the Manager Processor																																							
R	<table border="1"><tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">TOT_SAM</td></tr></table>																31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TOT_SAM				TOT_SAM			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TOT_SAM																																								
W	<table border="1"><tr><td> </td><td> </td></tr></table>																																																							
PO Reset	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																				
R	<table border="1"><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td colspan="4">TOT_SAM</td></tr></table>																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TOT_SAM				TOT_SAM																			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TOT_SAM																																								
W	<table border="1"><tr><td> </td><td> </td></tr></table>																																																							
PO Reset	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																				

**Figure B-25. RNG TRNG Total Samples Register - Format**

**Table B-29. RNG TRNG Total Samples Register - Field Descriptions**

Field	Description
31-20	Reserved. Always 0.
19-0 TOT_SAM	Total Samples. During Entropy generation, the total number of raw samples is counted. This count is useful in determining how often a sample is used during Von Neumann sampling. The count may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Sparse Bit Limit in register RTSBLIM, as defined in the previous section.

### B.2.2.9 RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)

The RNG TRNG Frequency Count Minimum Limit Register defines the minimum allowable count taken by the Entropy sample counter during each Entropy sample. During

any sample period, if the count is less than this programmed minimum, a Frequency Count Fail is flagged in RTMCTL[FCT\_FAIL] and an error is generated.

Offset	0xBASE_0618 (RTFRQMIN)	Access: Read/Write Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	FRQ_MIN
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	FRQ_MIN
W		

**Figure B-26. RNG TRNG Frequency Count Minimum Limit Register - Format**

**Table B-30. RNG TRNG Frequency Count Minimum Limit Register - Field Descriptions**

Field	Description
31-22	Reserved. Always 0.
21-0 FRQ_MIN	Frequency Count Minimum Limit. Defines the minimum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 0000h64 by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.10 RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)

The RNG TRNG Frequency Count Maximum Limit Register defines the maximum allowable count taken by the Entropy sample counter during each Entropy sample. During any sample period, if the count is greater than this programmed maximum, a Frequency Count Fail is flagged in RTMCTL[FCT\_FAIL] and an error is generated. Note that this address (061C) is used as RTFRQMAX only if RTMCTL[PRGM] is 1. If

RTMCTL[PRGM] is 0, this address is used as RTFRQCNT readback register, as described in the next section.

Offset	0xBASE_061C (RTFRQMAX)												Access: Read/Write Accessible only by the Manager Processor			
	(Accessible at this address when RTMCTL[PRGM] = 1)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	FRQ_MAX															

**Figure B-27. RNG TRNG Frequency Count Maximum Limit Register - Format**

**Table B-31. RNG TRNG Frequency Count Maximum Limit Register - Field Descriptions**

Field	Description
31-22	Reserved. Always 0.
21-0 FRQ_MAX	Frequency Counter Maximum Limit Defines the maximum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 000640h by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Frequency Count result in register RTFRQCNT, as defined in the following section.

### B.2.2.11 RNG TRNG Frequency Count Register (RTFRQCNT)

The RNG TRNG Frequency Count Register is a read-only register used to read the frequency counter within the TRNG entropy generator. It will read all zeroes unless RTMCTL[TRNG\_ACC] = 1. Note that this address (0xBASE\_061C) is used as RTFRQMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTFRQCNT readback register, as described here.

Offset	0xBASE_061C (RTFRQCNT)												Access: Read-Only Accessible only by the Manager Processor			
	(Accessible at this address when RTMCTL[PRGM] = 0)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	FRQ_CNT															
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

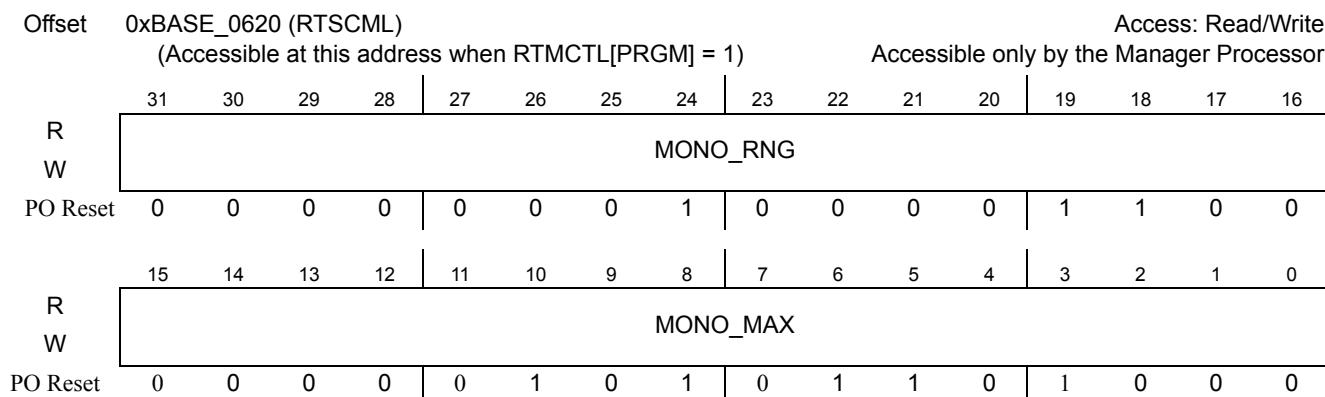
**Figure B-28. RNG TRNG Frequency Count Register - Format**

**Table B-32. RNG TRNG Frequency Count Register - Field Descriptions**

Field	Description
31-22	Reserved. Always 0.
21-0 FRQ_CNT	Frequency Count. If RTMCTL[TRNG_ACC] = 1, reads a sample frequency count taken during entropy generation. Requires RTMCTL[PRGM] = 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Poker Test Maximum Limit in register RTPKRMAX, as defined in the previous section.

### B.2.2.12 RNG TRNG Statistical Check Monobit Limit Register (RTSCML)

The RNG TRNG Statistical Check Monobit Limit Register defines the allowable maximum and minimum number of ones/zero detected during entropy generation. To pass the test, the number of ones/zeros generated must be less than the programmed maximum value, and the number of ones/zeros generated must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_0620) is used as RTSCML only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCMC readback register, as described in the next section.

**Figure B-29. RNG TRNG Statistical Check Monobit Limit Register - Format****Table B-33. RNG TRNG Statistical Check Monobit Limit Register - Field Descriptions**

Field	Description
31-16 MONO_RNG	Monobit Range. The number of ones/zeros detected during entropy generation must be greater than MONO_MAX - MONO_RNG, else a retry or error will occur. This register is cleared to 000112h (decimal 274) by writing the RTMCTL[RST_DEF] bit to 1.
15-0 MONO_MAX	Monobit Maximum Limit. Defines the maximum allowable count taken during entropy generation. The number of ones/zeros detected during entropy generation must be less than MONO_MAX, else a retry or error will occur. This register is cleared to 00056Bh (decimal 1387) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.13 RNG TRNG Statistical Check Monobit Count Register (RTSCMC)

The RNG TRNG Statistical Check Monobit Count Register is a read-only register used to read the final monobit count after entropy generation. This counter starts with the value in RTSCML[MONO\_MAX], and is decremented each time a one is sampled. Note that this address (0xBASE\_0620) is used as RTSCML if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCMC readback register, as described here.

Offset	0xBASE_0620 (RTSCMC) (Accessible at this address when RTMCTL[PRGM] = 0)																Access: Read-Only Accessible only by the Manager Processor																																				
R	<table border="1"><tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>																	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																					
W	<table border="1"><tr><td> </td><td> </td></tr></table>																																																				
PO Reset	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																				
R	<table border="1"><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td colspan="18">MONO_CNT</td></tr></table>																	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	MONO_CNT																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0																																				
MONO_CNT																																																					
W	<table border="1"><tr><td> </td><td> </td></tr></table>																																																				
PO Reset	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																				

Figure B-30. RNG TRNG Statistical Check Monobit Count Register - Format

Table B-34. RNG TRNG Statistical Check Monobit Count Register - Field Descriptions

Field	Description
31-16	Reserved. Always 0.
15-0 MONO_CNT	Monobit Count. Reads the final Monobit count after entropy generation. Requires RTMCTL[PRGM] = 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Statistical Check Monobit Limit in register RTSCML, as defined in the previous section.

### B.2.2.14 RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)

The RNG TRNG Statistical Check Run Length 1 Limit Register defines the allowable maximum and minimum number of runs of length 1 detected during entropy generation. To pass the test, the number of runs of length 1 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 1 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_0624) is used as

RTSCR1L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR1C readback register, as described in the next section.

Offset	0xBASE_0624 (RTSCR1L) (Accessible at this address when RTMCTL[PRGM] = 1)	Access: Read/Write Accessible only by the Manager Processor
	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
R	0	RUN1_RNG
W		
PO Reset	0 0 0 0   0 0 0 0   1 0 1 1   0 0 1 0	
	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
R	0	RUN1_MAX
W		
PO Reset	0 0 0 0   0 0 0 1   1 0 0 1   0 1 0 1	

**Figure B-31. RNG TRNG Statistical Check Run Length 1 Limit Register - Format**

**Table B-35. RNG TRNG Statistical Check Run Length 1 Limit Register - Field Descriptions**

Field	Description
31	Reserved. Always 0.
30-16 RUN1_RNG	Run Length 1 Range. The number of runs of length 1 (for both 0 and 1) detected during entropy generation must be greater than RUN1_MAX - RUN1_RNG, else a retry or error will occur. This register is cleared to 0102h (decimal 258) by writing the RTMCTL[RST_DEF] bit to 1.
15	Reserved. Always 0.
14-0 RUN1_MAX	Run Length 1 Maximum Limit. Defines the maximum allowable runs of length 1 (for both 0 and 1) detected during entropy generation. The number of runs of length 1 detected during entropy generation must be less than RUN1_MAX, else a retry or error will occur. This register is cleared to 01E5h (decimal 485) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.15 RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)

The RNG TRNG Statistical Check Run Length 1 Counters Register is a read-only register used to read the final Run Length 1 counts after entropy generation. These counters start with the value in RTSCR1L[RUN1\_MAX]. The R1\_1\_COUNT decrements each time a single one is sampled (preceded by a zero and followed by a zero). The R1\_0\_COUNT decrements each time a single zero is sampled (preceded by a one and followed by a one). Note that this address (0xBASE\_0624) is used as RTSCR1L if RTMCTL[PRGM] is 1. If

RTMCTL[PRGM] is 0, this address is used as RTSCR1C readback register, as described here.

Offset	0xBASE_0624 (RTSCR1C) (Accessible at this address when RTMCTL[PRGM] = 0)	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0   R1_1_COUNT	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0   R1_0_COUNT	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-32. RNG TRNG Statistical Check Runs Length 1 Count Register - Format**

**Table B-36. RNG TRNG Statistical Check Runs Length 1 Count Register - Field Descriptions**

Field	Description
31	Reserved. Always 0.
30-16 R1_1_COUNT	Runs of One, Length 1 Count. Reads the final Runs of Ones, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15	Reserved. Always 0.
14-0 R1_0_COUNT	Runs of Zero, Length 1 Count. Reads the final Runs of Zeroes, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.16 RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)

The RNG TRNG Statistical Check Run Length 2 Limit Register defines the allowable maximum and minimum number of runs of length 2 detected during entropy generation. To pass the test, the number of runs of length 2 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 2 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_0628) is used as

RTSCR2L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR2C readback register, as described in the next section.

Offset	0xBASE_0628 (RTSCR2L)	Access: Read/Write
	(Accessible at this address when RTMCTL[PRGM] = 1)	Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0	RUN2_RNG
W		
PO Reset	0 0 0 0   0 0 0 0   0 1 1 1   1 0 1 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0	RUN2_MAX
W		
PO Reset	0 0 0 0   0 0 0 0   1 1 0 1   1 1 0 0	

**Figure B-33. RNG TRNG Statistical Check Run Length 2 Limit Register - Format**

**Table B-37. RNG TRNG Statistical Check Run Length 2 Limit Register - Field Descriptions**

Field	Description
31-30	Reserved. Always 0.
29-16 RUN2_RNG	Run Length 2 Range The number of runs of length 2 (for both 0 and 1) detected during entropy generation must be greater than RUN2_MAX - RUN2_RNG, else a retry or error will occur. This register is cleared to 007Ah (decimal 122) by writing the RTMCTL[RST_DEF] bit to 1.
15-14	Reserved. Always 0.
13-0 RUN2_MAX	Run Length 2 Maximum Limit Defines the maximum allowable runs of length 2 (for both 0 and 1) detected during entropy generation. The number of runs of length 2 detected during entropy generation must be less than RUN2_MAX, else a retry or error will occur. This register is cleared to 00DCh (decimal 220) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.17 RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)

The RNG TRNG Statistical Check Run Length 2 Counters Register is a read-only register used to read the final Run Length 2 counts after entropy generation. These counters start with the value in RTSCR2L[RUN2\_MAX]. The R2\_1\_COUNT decrements each time two consecutive ones are sampled (preceded by a zero and followed by a zero). The R2\_0\_COUNT decrements each time two consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this address (0xBASE\_0628) is used as RTSCR2L if

RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR2C readback register, as described here.

Offset	0xBASE_0628 (RTSCR2C)														Access: Read-Only														
	(Accessible at this address when RTMCTL[PRGM] = 0)														Accessible only by the Manager Processor														
R2_1_COUNT																													
R2_0_COUNT																													
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														

**Figure B-34. RNG TRNG Statistical Check Runs Length 2 Count Register - Format**

**Table B-38. RNG TRNG Statistical Check Runs Length 2 Count Register - Field Descriptions**

Field	Description
31-30	Reserved. Always 0.
29-16 R2_1_COUNT	Runs of One Length 2 Count. Reads the final Runs of Ones, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-14	Reserved. Always 0.
13-0 R2_0_COUNT	Runs of Zero, Length 2 Count Reads the final Runs of Zeroes, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.18 RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L)

The RNG TRNG Statistical Check Run Length 3 Limit Register defines the allowable maximum and minimum number of runs of length 3 detected during entropy generation. To pass the test, the number of runs of length 3 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 3 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_062C) is used as

RTSCR3L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR3C readback register, as described in the next section.

Offset	0xBASE_062C (RTSCR3L)	Access: Read/Write
	(Accessible at this address when RTMCTL[PRGM] = 1)	Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0   0 0 0 0   0 1 0 1   1 0 0 0	RUN3_RNG
PO Reset	0 0 0 0   0 0 0 0   0 1 1 1   1 1 0 1	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   0 0 0 0   0 1 1 1   1 1 0 1	RUN3_MAX
PO Reset	0 0 0 0   0 0 0 0   0 1 1 1   1 1 0 1	

**Figure B-35. RNG TRNG Statistical Check Run Length 3 Limit Register - Format**

**Table B-39. RNG TRNG Statistical Check Run Length 3 Limit Register - Field Descriptions**

Field	Description
31-29	Reserved. Always 0.
28-16 RUN3_RNG	Run Length 3 Range The number of runs of length 3 (for both 0 and 1) detected during entropy generation must be greater than RUN3_MAX - RUN3_RNG, else a retry or error will occur. This register is cleared to 0058h (decimal 88) by writing the RTMCTL[RST_DEF] bit to 1.
15-13	Reserved. Always 0.
12-0 RUN3_MAX	Run Length 3 Maximum Limit Defines the maximum allowable runs of length 3 (for both 0 and 1) detected during entropy generation. The number of runs of length 3 detected during entropy generation must be less than RUN3_MAX, else a retry or error will occur. This register is cleared to 007Dh (decimal 125) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.19 RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)

The RNG TRNG Statistical Check Run Length 3 Counters Register is a read-only register used to read the final Run Length 3 counts after entropy generation. These counters start with the value in RTSCR3L[RUN3\_MAX]. The R3\_1\_COUNT decrements each time three consecutive ones are sampled (preceded by a zero and followed by a zero). The R3\_0\_COUNT decrements each time three consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this address (0xBASE\_062C) is used as RTSCR3L if

RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR3C readback register, as described here.

Offset	0xBASE_062C (RTSCR3C) (Accessible at this address when RTMCTL[PRGM] = 0)														Access: Read-Only Accessible only by the Manager Processor	
R3_1_COUNT																
R	0	0	0		27	26	25	24	23	22	21	20	19	18	17	16
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R3_0_COUNT																
R	0	0	0		11	10	9	8	7	6	5	4	3	2	1	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-36. RNG TRNG Statistical Check Runs Length 2 Count Register - Format**

**Table B-40. RNG TRNG Statistical Check Runs Length 2 Count Register - Field Descriptions**

Field	Description
31-29	Reserved. Always 0.
28-16 R3_1_COUNT	Runs of Ones, Length 3 Count. Reads the final Runs of Ones, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-13	Reserved. Always 0.
12-0 R3_0_COUNT	Runs of Zeroes, Length 3 Count. Reads the final Runs of Zeroes, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.20 RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L)

The RNG TRNG Statistical Check Run Length 4 Limit Register defines the allowable maximum and minimum number of runs of length 4 detected during entropy generation. To pass the test, the number of runs of length 4 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 4 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_0630) is used as

RTSCR4L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR4C readback register, as described in the next section.

Offset	0xBASE_0630 (RTSCR4L)	Access: Read/Write
	(Accessible at this address when RTMCTL[PRGM] = 1)	Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0 0 0   RUN4_RNG	
W		
PO Reset	0 0 0 0   0 0 0 0   0 1 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0 0 0   RUN4_MAX	
W		
PO Reset	0 0 0 0   0 0 0 0   0 1 0 0   1 0 1 1	

**Figure B-37. RNG TRNG Statistical Check Run Length 4 Limit Register - Format**

**Table B-41. RNG TRNG Statistical Check Run Length 4 Limit Register - Field Descriptions**

Field	Description
31-28	Reserved. Always 0.
27-16 RUN4_RNG	Run Length 4 Range The number of runs of length 4 (for both 0 and 1) detected during entropy generation must be greater than RUN4_MAX - RUN4_RNG, else a retry or error will occur. This register is cleared to 0040h (decimal 64) by writing the RTMCTL[RST_DEF] bit to 1.
15-12	Reserved. Always 0.
11-0 RUN4_MAX	Run Length 4 Maximum Limit Defines the maximum allowable runs of length 4 (for both 0 and 1) detected during entropy generation. The number of runs of length 4 detected during entropy generation must be less than RUN4_MAX, else a retry or error will occur. This register is cleared to 004Bh (decimal 75) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.21 RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C)

The RNG TRNG Statistical Check Run Length 4 Counters Register is a read-only register used to read the final Run Length 4 counts after entropy generation. These counters start with the value in RTSCR4L[RUN4\_MAX]. The R4\_1\_COUNT decrements each time four consecutive ones are sampled (preceded by a zero and followed by a zero). The R4\_0\_COUNT decrements each time four consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this address (0xBASE\_0630) is used as RTSCR4L if

RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR4C readback register, as described here.

Offset	0xBASE_0630 (RTSCR4C) (Accessible at this address when RTMCTL[PRGM] = 0)												Access: Read-Only Accessible only by the Manager Processor
R4_1_COUNT													
R	0	0	0	0	27	26	25	24	23	22	21	20	19
W													18
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	17
R4_0_COUNT													
R	0	0	0	0	11	10	9	8	7	6	5	4	3
W													2
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-38. RNG TRNG Statistical Check Runs Length 4 Count Register - Format**

**Table B-42. RNG TRNG Statistical Check Runs Length 4 Count Register - Field Descriptions**

Field	Description
31-28	Reserved. Always 0.
27-16 R4_1_COUNT	Runs of One, Length 4 Count Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-12	Reserved. Always 0.
11-0 R4_0_COUNT	Runs of Zero, Length 4 Count Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.2 RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L)

The RNG TRNG Statistical Check Run Length 5 Limit Register defines the allowable maximum and minimum number of runs of length 5 detected during entropy generation. To pass the test, the number of runs of length 5 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 5 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0xBASE\_0634) is used as

RTSCR5L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR5C readback register, as described in the next section.

Offset	0xBASE_0634 (RTSCR5L)	Access: Read/Write
	(Accessible at this address when RTMCTL[PRGM] = 1)	Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0 0 0   0   RUN5_RNG	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 1 0   1 1 1 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0 0 0   0   RUN5_MAX	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 1 0   1 1 1 1	

**Figure B-39. RNG TRNG Statistical Check Run Length 5 Limit Register - Format**

**Table B-43. RNG TRNG Statistical Check Run Length 5 Limit Register - Field Descriptions**

Field	Description
31-27	Reserved. Always 0.
26-16 RUN5_RNG	Run Length 5 Range The number of runs of length 5 (for both 0 and 1) detected during entropy generation must be greater than RUN5_MAX - RUN5_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1.
15-11	Reserved. Always 0.
10-0 RUN5_MAX	Run Length 5 Maximum Limit Defines the maximum allowable runs of length 5 (for both 0 and 1) detected during entropy generation. The number of runs of length 5 detected during entropy generation must be less than RUN5_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.23 RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C)

The RNG TRNG Statistical Check Run Length 5 Counters Register is a read-only register used to read the final Run Length 5 counts after entropy generation. These counters start with the value in RTSCR5L[RUN5\_MAX]. The R5\_1\_COUNT decrements each time five consecutive ones are sampled (preceded by a zero and followed by a zero). The R5\_0\_COUNT decrements each time five consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this address (0xBASE\_0634) is used as RTSCR5L if

RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR5C readback register, as described here.

Offset	0xBASE_0634 (RTSCR5C) (Accessible at this address when RTMCTL[PRGM] = 0)												Access: Read-Only Accessible only by the Manager Processor																			
R																																
W																																
PO Reset																																
R																																
W																																
PO Reset																																
R																																
W																																
R																																
W																																
PO Reset																																

**Figure B-40. RNG TRNG Statistical Check Runs Length 5 Count Register - Format**

**Table B-44. RNG TRNG Statistical Check Runs Length 5 Count Register - Field Descriptions**

Field	Description
31-27	Reserved. Always 0.
26-16 R5_1_COUNT	Runs of One, Length 5 Count Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-11	Reserved. Always 0.
10-0 R5_0_COUNT	Runs of Zero, Length 5 Count Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.24 RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL)

The RNG TRNG Statistical Check Run Length 6+ Limit Register defines the allowable maximum and minimum number of runs of length 6 or more detected during entropy generation. To pass the test, the number of runs of length 6 or more (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 6 or more must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address

( $_{0x\text{BASE\_0638}}$ ) is used as RTSCR6PL only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR6PC readback register, as described in the next section.

Offset	0x $\text{BASE\_0638}$ (RTSCR6PL) (Accessible at this address when RTMCTL[PRGM] = 1)												Access: Read/Write Accessible only by the Manager Processor															
R	31	30	29	28	27	26	25	24	23	22	21	20	RUN6P_RNG															
W																												
PO Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	1	1	1	0							
W													RUN6P_MAX															
PO Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1							

Figure B-41. RNG TRNG Statistical Check Run Length 6+ Limit Register - Format

Table B-45. RNG TRNG Statistical Check Run Length 6+ Limit Register - Field Descriptions

Field	Description
31-27	Reserved. Always 0.
26-16 RUN6P_RNG	Run Length 6+ Range The number of runs of length 6 or more (for both 0 and 1) detected during entropy generation must be greater than RUN6P_MAX - RUN6P_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1.
15-11	Reserved. Always 0.
10-0 RUN6P_MAX	Run Length 6+ Maximum Limit Defines the maximum allowable runs of length 6 or more (for both 0 and 1) detected during entropy generation. The number of runs of length 6 or more detected during entropy generation must be less than RUN6P_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1.

### B.2.2.25 RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC)

The RNG TRNG Statistical Check Run Length 6+ Counters Register is a read-only register used to read the final Run Length 6+ counts after entropy generation. These counters start with the value in RTSCR6PL[RUN6P\_MAX]. The R6P\_1\_COUNT decrements each time six or more consecutive ones are sampled (preceded by a zero and followed by a zero). The R6P\_0\_COUNT decrements each time six or more consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this address ( $_{0x\text{BASE\_0638}}$ ) is used as

RTSCR6PL if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR6PC readback register, as described here.

Offset	0xBASE_0638 (RTSCR6PC) (Accessible at this address when RTMCTL[PRGM] = 0)																Access: Read-Only Accessible only by the Manager Processor
R																	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset																	
R																	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset																	
R																	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset																	

**Figure B-42. RNG TRNG Statistical Check Runs Length 6+ Count Register - Format**

**Table B-46. RNG TRNG Statistical Check Runs Length 6+ Count Register - Field Descriptions**

Field	Description
31-27	Reserved. Always 0.
26-16 R6P_1_COUNT	Runs of One, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-11	Reserved. Always 0.
10-0 R6P_0_COUNT	Runs of Zero, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0.

### B.2.2.26 RNG TRNG Status Register (RTSTATUS)

Various statistical tests are run as a normal part of the TRNG's entropy generation process. The least-significant 16 bits of the RTSTATUS register reflect the result of each of these tests. The status of these bits will be valid when the TRNG has finished its entropy generation process. Software can determine when this occurs by polling the ENT\_VAL bit in the RNG TRNG Miscellaneous Control Register.

Note that there is a very small probability that a statistical test will fail even though the TRNG is operating properly. If this happens the TRNG will automatically retry the entire entropy generation process, including running all the statistical tests. The value in RETRY\_COUNT is decremented each time an entropy generation retry occurs. If a statistical check fails when the retry count is nonzero, a retry is initiated. But if a statistical check fails when the retry count is zero, an error is generated by the RNG. By default

RETRY\_COUNT is initialized to 1, but software can increase the retry count by writing to the RTY\_CNT field in the RTSCMISC register (see Section [B.2.2.2 on page B-478](#)).

All 0s will be returned if this register address is read while the RNG is in Program Mode (see PRGM field in RTMCTL register (see Section [B.2.2.1 on page B-476](#)). If this register is read while the RNG is in Run Mode the value returned will be formatted as follows.

Offset	0xBASE_063C (RTSTATUS)												Access: Read-Only			
	(Accessible at this address when RTMCTL[PRGM] = 0)												Accessible only by the Manager Processor			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MBTF	PTF	LRTF	SBTF	6PBR1TF	6PBR0TF	5BR1TF	5BR0TF	4BR1TF	4BR0TF	3BR1TF	3BR0TF	2BR1TF	2BR0TF	1BR1TF	1BR0TF
W					0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-43. RNG TRNG Status Register - Format**

**Table B-47. RNG TRNG Status Register - Field Descriptions**

Field	Description
31-20	Reserved. Always 0.
19-16 RETRY_COUNT	RETRY COUNT This represents the current number of entropy generation retries left before a statistical test failure will cause the RNG to generate an error condition.
15 MBTF	Mono Bit Test Fail. If MBTF=1, the Mono Bit Test has failed.
14 PTF	Poker Test Fail. If PTF=1, the Poker Test has failed.
13 LRTF	Long Run Test Fail. If LRTF=1, the Long Run Test has failed.
12 SBTF	Sparse Bit Test Fail. If SBTF=1, the Sparse Bit Test has failed.
11 6PBR1TF	6 Plus Bit Run, Sampling 1s,Test Fail. If 6PBR1TF=1, the 6 Plus Bit Run, Sampling 1s Test has failed.
10 6PBR0TF	6 Plus Bit Run, Sampling 0s,Test Fail. If 6PBR0TF=1, the 6 Plus Bit Run, Sampling 0s Test has failed.

**Table B-47. RNG TRNG Status Register - Field Descriptions**

Field	Description
9 5BR1TF	5-Bit Run, Sampling 1s, Test Fail. If 5BR1TF=1, the 5-Bit Run, Sampling 1s Test has failed.
8 5BR0TF	5-Bit Run, Sampling 0s, Test Fail. If 5BR0TF=1, the 5-Bit Run, Sampling 0s Test has failed.
7 4BR1TF	4-Bit Run, Sampling 1s, Test Fail. If 4BR1TF=1, the 4-Bit Run, Sampling 1s Test has failed.
6 4BR0TF	4-Bit Run, Sampling 0s, Test Fail. If 4BR0TF=1, the 4-Bit Run, Sampling 0s Test has failed.
5 3BR1TF	3-Bit Run, Sampling 1s, Test Fail. If 3BR1TF=1, the 3-Bit Run, Sampling 1s Test has failed.
4 3BR01TF	3-Bit Run, Sampling 0s, Test Fail. If 3BR01TF=1, the 3-Bit Run, Sampling 0s Test has failed.
3 2BR1TF	2-Bit Run, Sampling 1s, Test Fail. If 2BR1TF=1, the 2-Bit Run, Sampling 1s Test has failed.
2 2BR0TF	2-Bit Run, Sampling 0s, Test Fail. If 2BR0TF=1, the 2-Bit Run, Sampling 0s Test has failed.
1 1BR1TF	1-Bit Run, Sampling 1s, Test Fail. If 1BR1TF=1, the 1-Bit Run, Sampling 1s Test has failed.
0 1BR0TF	1-Bit Run, Sampling 0s, Test Fail. If 1BR0TF=1, the 1-Bit Run, Sampling 0s Test has failed.

### B.2.2.27 RNG TRNG Entropy Read Registers (RTENT0 - RTENT11)

The RNG TRNG can be programmed to generate an entropy value that is readable via the SkyBlue bus. To do this, set the RTMCTL[TRNG\_ACC] bit to 1. Once the entropy value has been generated, the RTMCTL[ENT\_VAL] bit will be set to 1. At this point, RTENT0 through RTENT11 may be read to retrieve the 384-bit entropy value. Note that once RTENT11 is read, the entropy value will be cleared and a new value will begin generation, so it is important that RTENT11 be read last. Also note that the entropy value read from the RTENT0-RTENT11 registers will never be used by the CAAM for any purpose other than to be read via these registers. Any entropy value used for any security function cannot be read. These registers are readable only when RTMCTL[PRGM] = 0 (Run Mode), RTMCTL[TRNG\_ACC] = 1 (TRNG access mode) and RTMCTL[ENT\_VAL] = 1, otherwise zeroes will be read.

Offset	0xBASE_0640 (RTENT<i>), offset 4, range 0 .. 11	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	ENT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	ENT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-44. RNG TRNG Entropy Read Registers - Format****Table B-48. RNG TRNG Entropy Read Registers - Field Descriptions**

Field	Description
31-0 ENT	Entropy Value. Will be non-zero only if RTMCTL[PRGM] = 0 (Run Mode) and RTMCTL[ENT_VAL] = 1 (Entropy Valid). The most significant bits of the entropy are read from address 0640, and the least significant bits are read from address 0xBASE_066C. Note that reading address 0xBASE_066C also clears the entire entropy value, and starts a new entropy generation.

### B.2.2.28 RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)

The RNG TRNG Statistical Check Poker Count 1 and 0 Register is a read-only register used to read the final Poker test counts of 1h and 0h patterns. The Poker 0h Count increments each time a nibble of sample data is found to be 0h. The Poker 1h Count increments each time a nibble of sample data is found to be 1h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

Offset	0xBASE_0680 (RTPKRCNT10)	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	PKR_1_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	PKR_0_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-45. RNG TRNG Statistical Check Poker Count 1 and 0 Register - Format**

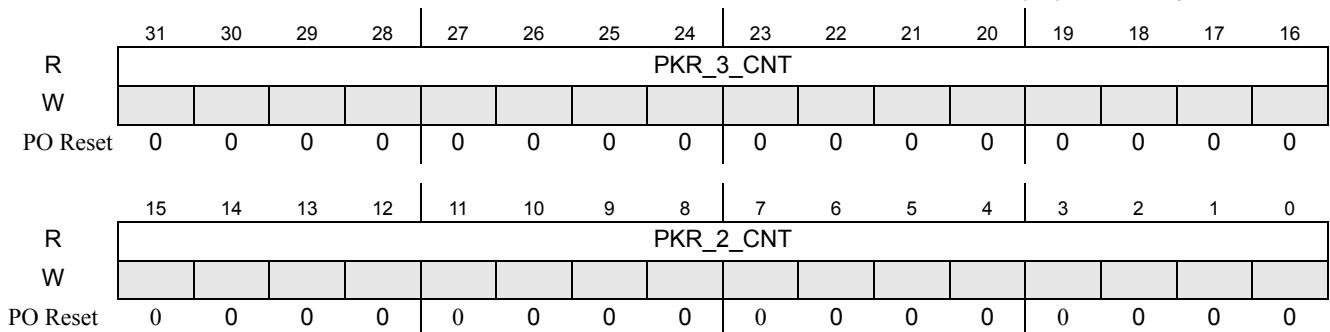
**Table B-49. RNG TRNG Statistical Check Poker Count 1 and 0 Register - Field Descriptions**

Field	Description
31-16 PKR_1_CNT	Poker 1h Count. Total number of nibbles of sample data which were found to be 1h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_0_CNT	Poker 0h Count. Total number of nibbles of sample data which were found to be 0h. Requires RTMCTL[PRGM] = 0.

**B.2.2.29 RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)**

The RNG TRNG Statistical Check Poker Count 3 and 2 Register is a read-only register used to read the final Poker test counts of 3h and 2h patterns. The Poker 2h Count increments each time a nibble of sample data is found to be 2h. The Poker 3h Count increments each time a nibble of sample data is found to be 3h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

Offset 0xBASE\_0684 (RTPKRCNT32)

Access: Read-Only  
Accessible only by the Manager Processor**Figure B-46. RNG TRNG Statistical Check Poker Count 3 and 2 Register - Format****Table B-50. RNG TRNG Statistical Check Poker Count 3 and 2 Register - Field Descriptions**

Field	Description
31-16 PKR_3_CNT	Poker 3h Count. Total number of nibbles of sample data which were found to be 3h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_2_CNT	Poker 2h Count. Total number of nibbles of sample data which were found to be 2h. Requires RTMCTL[PRGM] = 0.

### **B.2.2.30 RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)**

The RNG TRNG Statistical Check Poker Count 5 and 4 Register is a read-only register used to read the final Poker test counts of 5h and 4h patterns. The Poker 4h Count increments each time a nibble of sample data is found to be 4h. The Poker 5h Count increments each time a nibble of sample data is found to be 5h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

**Figure B-47. RNG TRNG Statistical Check Poker Count 5 and 4 Register - Format**

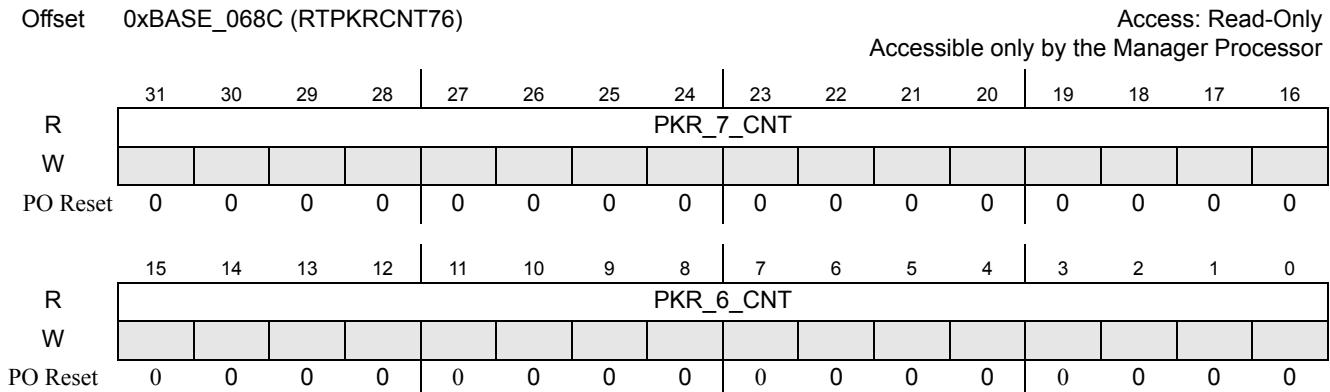
**Table B-51. RNG TRNG Statistical Check Poker Count 5 and 4 Register - Field Descriptions**

Field	Description
31-16 PKR_5_CNT	Poker 5h Count. Total number of nibbles of sample data which were found to be 5h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_4_CNT	Poker 4h Count. Total number of nibbles of sample data which were found to be 4h. Requires RTMCTL[PRGM] = 0.

### **B.2.2.31 RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)**

The RNG TRNG Statistical Check Poker Count 7 and 6 Register is a read-only register used to read the final Poker test counts of 7h and 6h patterns. The Poker 6h Count increments each time a nibble of sample data is found to be 6h. The Poker 7h Count increments each

time a nibble of sample data is found to be 7h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.



**Figure B-48. RNG TRNG Statistical Check Poker Count 7 and 6 Register - Format**

**Table B-52. RNG TRNG Statistical Check Poker Count 7 and 6 Register - Field Descriptions**

Field	Description
31-16 PKR_7_CNT	Poker 7h Count. Total number of nibbles of sample data which were found to be 7h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_6_CNT	Poker 6h Count. Total number of nibbles of sample data which were found to be 6h. Requires RTMCTL[PRGM] = 0.

### B.2.2.32 RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)

The RNG TRNG Statistical Check Poker Count 9 and 8 Register is a read-only register used to read the final Poker test counts of 9h and 8h patterns. The Poker 8h Count increments each time a nibble of sample data is found to be 8h. The Poker 9h Count

increments each time a nibble of sample data is found to be 9h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

Offset	0xBASE_0690 (RTPKRCNT98)	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	PKR_9_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	PKR_8_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-49. RNG TRNG Statistical Check Poker Count 9 and 8 Register - Format**

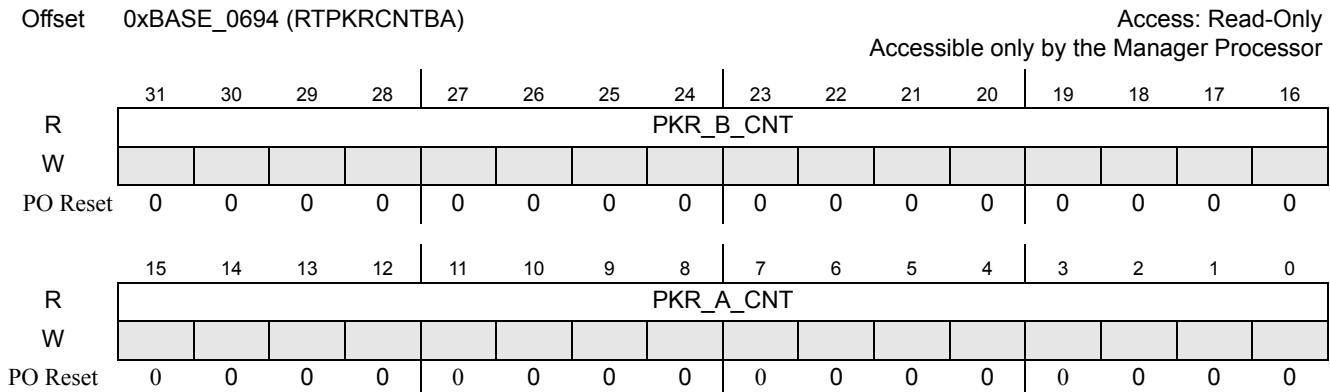
**Table B-53. RNG TRNG Statistical Check Poker Count 9 and 8 Register - Field Descriptions**

Field	Description
31-16 PKR_9_CNT	Poker 9h Count. Total number of nibbles of sample data which were found to be 9h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_8_CNT	Poker 8h Count. Total number of nibbles of sample data which were found to be 8h. Requires RTMCTL[PRGM] = 0.

### B.2.2.33 RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)

The RNG TRNG Statistical Check Poker Count B and A Register is a read-only register used to read the final Poker test counts of Bh and Ah patterns. The Poker Ah Count increments each time a nibble of sample data is found to be Ah. The Poker Bh Count

increments each time a nibble of sample data is found to be Bh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.



**Figure B-50. RNG TRNG Statistical Check Poker Count B and A Register - Format**

**Table B-54. RNG TRNG Statistical Check Poker Count B and A Register - Field Descriptions**

Field	Description
31-16 PKR_B_CNT	Poker Bh Count. Total number of nibbles of sample data which were found to be Bh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_A_CNT	Poker Ah Count. Total number of nibbles of sample data which were found to be Ah. Requires RTMCTL[PRGM] = 0.

### B.2.2.34 RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)

The RNG TRNG Statistical Check Poker Count D and C Register is a read-only register used to read the final Poker test counts of Dh and Ch patterns. The Poker Ch Count increments each time a nibble of sample data is found to be Ch. The Poker Dh Count

increments each time a nibble of sample data is found to be Dh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

Offset	0xBASE_0698 (RTPKRCNTDC)	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	PKR_D_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	PKR_C_CNT
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-51. RNG TRNG Statistical Check Poker Count D and C Register - Format**

**Table B-55. RNG TRNG Statistical Check Poker Count D and C Register - Field Descriptions**

Field	Description
31-16 PKR_D_CNT	Poker Dh Count. Total number of nibbles of sample data which were found to be Dh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_C_CNT	Poker Ch Count. Total number of nibbles of sample data which were found to be Ch. Requires RTMCTL[PRGM] = 0.

### B.2.2.35 RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)

The RNG TRNG Statistical Check Poker Count F and E Register is a read-only register used to read the final Poker test counts of Fh and Eh patterns. The Poker Eh Count increments each time a nibble of sample data is found to be Eh. The Poker Fh Count

## CAAM Register Descriptions

increments each time a nibble of sample data is found to be Fh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

Offset	0xBASE_069C (RTPKRCNTFE)																Access: Read-Only Accessible only by the Manager Processor
PKR_F_CNT																	
R																	
W																	
PO Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																
PKR_E_CNT																	
R																	
W																	
PO Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																

**Figure B-52. RNG TRNG Statistical Check Poker Count F and E Register - Format**

**Table B-56. RNG TRNG Statistical Check Poker Count F and E Register - Field Descriptions**

Field	Description
31-16 PKR_F_CNT	Poker Fh Count. Total number of nibbles of sample data which were found to be Fh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_E_CNT	Poker Eh Count. Total number of nibbles of sample data which were found to be Eh. Requires RTMCTL[PRGM] = 0.

### B.2.2.36 RNG DRNG Status Register (RDSTA)

The RNG DRNG Status Register shows the current status of the DRNG portion of the RNG.

Offset	0xBASE_06C0 (RDSTA)																Access: Read-only Accessible only by the Manager Processor
ERRCODE																	
R																	
W																	
PO Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																
IF0																	
R																	
W																	
PO Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																

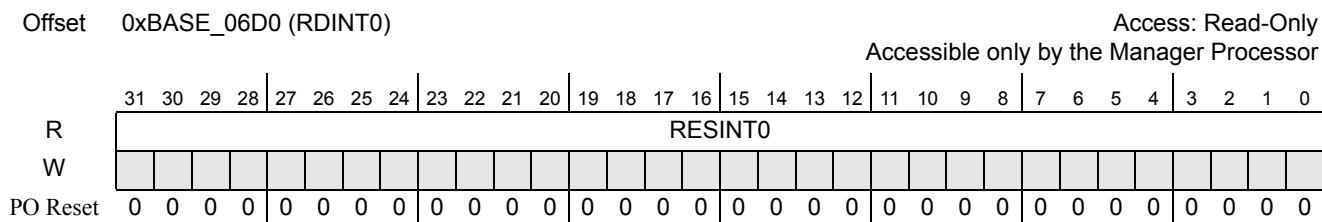
**Figure B-53. RNG DRNG Status Register**

**Table B-57. RNG DRNG Status Register - Field Descriptions**

Field	Description
31 SKVT	Secure Key Valid Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a test (deterministic) instance.
30 SKVN	Secure Key Valid Non-Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a non-test (non-deterministic) instance.
29-21	Reserved
20 CE	Catastrophic Error. A catastrophic error will occur when the RNG gets a hardware error while requesting new entropy and the current State Handle is instantiated as a non-test (non-deterministic) instance.
19-16 ERRCODE	Error Code. These bits represent the current error in the RNG.
15-12	Reserved.
11	Reserved.
10	Reserved.
9 TF1	Test Flag State Handle 1. State handle 1 has been instantiated as a test (deterministic) instance.
8 TF0	Test Flag State Handle 0. State handle 0 has been instantiated as a test (deterministic) instance.
7	Reserved.
6	Reserved.
5 PR1	Prediction Resistance Flag State Handle 1. State Handle 1 has been instantiated to support prediction resistance.
4 PR0	Prediction Resistance Flag State Handle 0. State Handle 0 has been instantiated to support prediction resistance.
3	Reserved.
2	Reserved.
1 IF1	Instantiated Flag State Handle 1. State Handle 1 has been instantiated.
0 IF0	Instantiated Flag State Handle 0. State Handle 0 has been instantiated.

### B.2.2.37 RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)

The RNG DRNG State Handle 0 Reseed Interval Register shows the current value of the reseed interval for State Handle 0. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT0 will be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT0. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see AS RNG OPERATION command settings).



**Figure B-54. RNG DRNG State Handle 0 Reseed Interval Register - Format**

**Table B-58. RNG DRNG State Handle 0 Reseed Interval Register - Field Descriptions**

Field	Description
31-0 RESINT0	RESINT0. This read-only register holds the Reseed Interval for State Handle 0.

### B.2.2.38 RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)

The RNG DRNG State Handle 1 Reseed Interval Register shows the current value of the reseed interval for State Handle 1. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT1 will be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT1. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see AS RNG OPERATION command settings).

a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see AS RNG OPERATION command settings).

Offset	0xBASE_06D4 (RDINT1)	Access: Read-Only Accessible only by the Manager Processor
	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16   15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
R	RESINT1	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-55. RNG DRNG State Handle 1 Reseed Interval Register - Format**

**Table B-59. RNG DRNG State Handle 1 Reseed Interval Register (RDINT1) - Field Descriptions**

Field	Description
31-0 RESINT1	RESINT1. This read-only register holds the Reseed Interval for State Handle 1.

### B.2.2.39 RNG DRNG Hash Control Register (RDHCNTL)

The RNG DRNG Hash Control Register is used to gain control of the SHA-256 hashing engine that is internal to the RNG. Once Hashing test mode is initialized then the user can begin the hashing operation and poll for the done bit.

Offset	0xBASE_06E0 (RDHCNTL)	Access: Read/Write Accessible only by the Manager Processor
	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
R	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
R	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-56. RNG DRNG Hash Control Register - Format**

**Table B-60. RNG DRNG Hash Control Register - Field Descriptions**

Field	Description
31-5	Reserved.
4 HTC	Hashing Test Mode Clear. Writing this bit will take the RNG out of hashing test mode.
3 HTM	Hashing Test Mode. Writing this bit will put RNG in Hashing Test Mode.
2 HI	Hashing Initialize. Writing to this bit will initialize the Hashing Engine.
1 HB	Hashing Begin. Writing this bit will cause the Hashing Engine to begin Hashing.
0 HD	Hashing Done. This bit asserts when the hashing engine is done.

**B.2.2.40 RNG DRNG Hash Digest Register (RDHDIG)**

The RNG DRNG Hash Digest Register allows user access to the eight 32-bit message digest registers of the SHA-256 hashing engine that is internal to the RNG. All eight registers are read in order from most-significant bits to least-significant bits by reading this address eight times. These registers are only readable while in Hashing Test Mode and when the Hashing Engine is done.

Offset	0xBASE_06E4 (RDHDIG)	Access: Read-Only Accessible only by the Manager Processor
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	HASHMD
W	[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	HASHMD
W	[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-57. RNG DRNG Hash Digest Register - Format****Table B-61. RNG DRNG Hash Digest Register - Field Descriptions**

Field	Description
31-0 HASHMD	HASHMD. Hashing Message Digest Register. This register needs to be read 8 times to retrieve the entire message digest.

### B.2.2.41 RNG DRNG Hash Buffer (RDHBUF)

The RNG DRNG Hash buffer allows access to the SHA-256 hashing engine that is internal to the RNG for the purpose of conformance testing. To fill the buffer this register must be written 16 times at this address. This register is writable only while the RNG is in Hashing Test mode. This mode can be selected via the RNG DRNG Hash Control Register.

Offset 0xBASE_06E8 (RDHBUF)																									Access: Write-Only Accessible only by the Manager Processor		

## CAAM Register Descriptions

most-significant bits in HTx\_JD\_ADDR\_MS, and the least-significant 32 bits in HTx\_JD\_ADDR\_LS.

CAAM implements holding tanks. The JD\_ADDR\_MS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C00 (HT0_JD_ADDR_MS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-59. HTx\_JD\_ADDR\_MS Register - Format**

CAAM implements holding tanks. The JD\_ADDR\_LS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C04 (HT0_JD_ADDR_LS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 JD_ADDR_LS	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 JD_ADDR_LS	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-60. HTx\_JD\_ADDR\_LS Register - Format**

**Table B-63. HTx\_JD\_ADDR Register - Field Descriptions**

Field	Description
Most-significant portion of register	
31-0	Reserved.
Least-significant portion of register	
31-0 JD_ADDR_LS	Job Descriptor Address.

### B.2.3.2 Holding Tank Shared Descriptor Address Register (HTx\_SD\_ADDR)

The HTx\_SD\_ADDR register holds the address of a Shared Descriptor that is in a “holding tank” waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution via a Job Ring. In versions of CAAM that implement 36-bit or 40-bit addresses, the HTx\_SD\_ADDR register is accessed as two 32-bit registers, with the most-significant bits in HTx\_SD\_ADDR\_MS, and the least-significant 32 bits in HTx\_SD\_ADDR\_LS.

CAAM implements holding tanks. The SD\_ADDR\_MS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C08 (HT0_SD_ADDR_MS)	Access:
		Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-61. HTx\_SD\_ADDR\_MS Register - Format**

CAAM implements holding tanks. The SD\_ADDR\_LS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C0C (HT0_SD_ADDR_LS)	Access:
		Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	SD_ADDR_LS
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	SD_ADDR_LS
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-62. HTx\_SD\_ADDR\_LS Register - Format**

**Table B-64. HTx\_SD\_ADDR Register - Field Descriptions**

Field	Description
Most-significant portion of register	
31-0	Reserved.
Least-significant portion of register	
31-0 SD_ADDR_LS	Shared Descriptor Address.

### B.2.3.3 Holding Tank Job Queue Control Debug Register (HTx\_JQCTRL)

The HTx\_JQCTRL register holds the control information for a descriptor that is in a “holding tank” waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The HTx\_JQCTRL register is accessed as two 32-bit registers. The most-significant half of HTx\_JQCTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

CAAM implements holding tanks. The JQ\_CTRL\_MS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C10 (HT0_JQCTRL_MS)	Access: Read-Only																																				
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16																																					
	<table border="1"> <tr><td>0</td><td>0</td><td>WHL</td><td>FOUR</td><td>ILE</td><td colspan="4">Reserved</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	0	0	WHL	FOUR	ILE	Reserved				0	0	0	0	0	0	0	0																				
0	0	WHL	FOUR	ILE	Reserved				0	0	0	0	0	0	0	0																						
W																																						
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0																																					
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0																																					
	<table border="1"> <tr><td>AMTD</td><td>JDMS</td><td>0</td><td>0</td><td>0</td><td>SRC</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">ID</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	AMTD	JDMS	0	0	0	SRC	0	0	0	0	ID				0	0	0	0																			
AMTD	JDMS	0	0	0	SRC	0	0	0	0	ID				0	0	0	0																					
W																																						
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0																																					

**Figure B-63. HTx\_JQCTRL\_MS Register - Format**

CAAM implements holding tanks. The JTCTRL\_LS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C14 (HT0_JQCTRL_LS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0 0 0   Non-SEQ_MID	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0 0 0   SEQ_MID	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-64. HTx\_JQCTRL\_LS Register - Format

Table B-65. HTx\_JQCTRL Register - Field Descriptions

Field	Description
Most-significant portion of register	
31-30	Reserved.
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor will be given to DECO by the Job Queue Controller.
28 FOUR	Four Words. Job Queue Controller will pass at least 4 words of the Descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. ILE = 0: No byte-swapping is performed for data transferred to or from the Descriptor Buffer. ILE = 1: Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.
26-24	Reserved.
23-16	Reserved.
15 AMTD	Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRMID Register.
14 JDMS	Job Descriptor MID Select. Determines whether the SEQ MID or the Non-SEQ MID is asserted when reading the Job Descriptor from memory. 1 = SEQ MID 0 = Non-SEQ MID
13-11	Reserved.

Field	Description
10-8 SRC	Job Source. Source of the job. Determines which set of endian configuration bits that the DMA should use for transactions. It is illegal for the SRC field to have a value other than that of a Job Ring when running descriptors via the IP bus (i.e. under the direct control of software). 0b000: Job Ring 0 0b001: Job Ring 1 0b010: Reserved 0b011: Reserved 0b100: Reserved 0b101: Reserved 0b110: Reserved 0b111: Reserved
7-4	Reserved.
3-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.
Least-significant portion of register	
31-28	Reserved.
27-16 NON_SEQ_ MID	Non-SEQ MID. This is the value that will be used whenever the Non-SEQ MID is asserted on the AXI Master interface when this descriptor is fetched and executed.
15-12	Reserved.
11-0 SEQ_MID	SEQ MID. This is the value that will be used whenever the SEQ MID is asserted on the AXI Master interface when this descriptor is fetched and executed.

### B.2.3.4 Holding Tank Status Register (HTx\_STATUS)

The HTx\_STATUS register holds the status information for a Job Descriptor holding tank. The register is intended to be used when debugging descriptor execution.

CAAM implements holding tanks. The STATUS registers for these holding tanks are accessible at the addresses shown below.

Offset	0xBASE_0C1C (HT0_STATUS)																Access: Read-Only
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	BC	IN_USE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	BC	IN_USE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PEND_0	
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-65. HTx\_STATUS Register - Format

Table B-66. HTx\_STATUS Register - Field Descriptions

Field	Description
31 BC	Been Changed. When using the Holding Tank debug registers, the Holding Tank Job Descriptor Address register should be the first register that is read. The BC ("Been Changed") bit is cleared when the Holding Tank Job Descriptor Address register is read. If data in the holding tanks changes after that time but before the HT Status register is read, the "Been Changed" bit is set. This indicates that the data read from some of the HT debug registers may be inconsistent with data read from other HT debug registers. In this case the HT debug registers should be reread, starting with the Holding Tank Job Descriptor Address register.
30 IN_USE	In Use. The "In use" bit is set when the HT contains some or all of the information for a job that has not yet been sent or not yet completely sent to a DECO.
29-16	Reserved.
15-1	Reserved.
0 PEND_0	Pending for DECO 0. The PEND_0 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 0. It is possible for more than one pending bit for the holding tank to be set at the same time.

### B.2.3.5 Job Ring Address-Array Address Registers (JRAAAx)

The JRAAA registers are intended to be used when debugging descriptor execution via a Job Ring. As discussed in job scheduling, the job controller buffers up to four<sup>1</sup> job descriptors from one job ring before servicing the next job ring in round-robin fashion. The

1. Eight JRAAA register addresses are defined, but only the first four JRAAA registers are implemented in this version of CAAM.

## CAAM Register Descriptions

JRAAA registers are used to buffer the addresses of those four job descriptors. In versions of CAAM that implement 36-bit or 40-bit addresses, the address is read as two 32-bit words. The most-significant bits are in the MS register and the 32 least-significant bits are in the LS register.

Offset	0xBASE_0D70 (JRAAA0_MS) 0xBASE_0D78 (JRAAA1_MS) 0xBASE_0D80 (JRAAA2_MS) 0xBASE_0D88 (JRAAA3_MS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-66. Job Ring Address-Array Address Registers\_MS - Format**

Offset	0xBASE_0D74 (JRAAA0_LS) 0xBASE_0D7C (JRAAA1_LS) 0xBASE_0D84 (JRAAA2_LS) 0xBASE_0D8C (JRAAA3_LS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 JD_ADDR_LS	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 JD_ADDR_LS	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-67. Job Ring Address-Array Address Registers\_LS - Format**

**Table B-67. Job Ring Address-Array Address Registers - Field Descriptions**

Field	Description
Most-significant portion of register	
31-0	Reserved.
Least-significant portion of register	
31-0 JD_ADDR_LS	Job Descriptor Address.

### B.2.3.6 Job Ring Address-Array Valid/Source Register (JRAAVS)

The Job Ring Address-Array Valid/Source register indicates the validity and source of the job descriptor addresses stored in the Job Ring Address-Array Address Registers. The register is intended to be used when debugging descriptor execution via a job ring.

Offset	0xBASE_0DB4 (JRAAVS)																Access:
																	Read-Only
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	V3	0	SRC3		V2	0	SRC2		V1	0	SRC1		V0	0	SRC0		
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-68. Job Ring Address-Array Valid/Source Register - Format

Table B-68. Job Ring Address-Array Valid/Source Register - Field Descriptions

Field	Description
31-16	Reserved.
15 V3	Valid 3. When V3=1, Job Ring Address-Array Address Register 3 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank.
14	Reserved.
13-12 SRC3	Source 3. When V3=1, SRC3 indicates the number of the Job Ring from which the Job Descriptor address held in Ring Address-Array Address Register 3 was read.
11 V2	Valid 2. When V2=1, Job Ring Address-Array Address Register 2 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank.
10	Reserved.
9-8 SRC2	Source 2. When V2=1, SRC2 indicates the number of the Job Ring from which the Job Descriptor address held in Ring Address-Array Address Register 2 was read.
7 V1	Valid 1. When V1=1, Job Ring Address-Array Address Register 1 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank.
6	Reserved.
5-4 SRC1	Source 1. When V1=1, SRC1 indicates the number of the Job Ring from which the Job Descriptor address held in Ring Address-Array Address Register 1 was read.

Field	Description
3 V0	Valid 0. When V0=0, Job Ring Address-Array Address Register 0 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank.
2	Reserved.
1-0 SRC0	Source 0. When V0=1, SRC0 indicates the number of the Job Ring from which the Job Descriptor address held in Ring Address-Array Address Register 0 was read.

### B.2.3.7 Job Ring Job IDs in Use Register (JRJIDU)

The Job Ring Job IDs in Use register indicates which of the Job IDs tracked by the Job Controller are currently in use (i.e. identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring). The register is intended to be used when debugging descriptor execution via a Job Ring. The most-significant half of JRJIDU contains a bit for each of the Job IDs, indicating whether that Job ID is currently in use. The least-significant half of JRJIDU contains the BC (Been Changed) bit, which is used to ensure that consistent data has been read from the Job Ring Address\_Array Registers.

Offset 0xBASE\_0DB8 (JRJIDU\_MS) Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JID15	JID14	JID13	JID12	JID11	JID10	JID09	JID08	JID07	JID06	JID05	JID04	JID03	JID02	JID01	JID00
R	JID15	JID14	JID13	JID12	JID11	JID10	JID09	JID08	JID07	JID06	JID05	JID04	JID03	JID02	JID01	JID00
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-69. Job Ring Job IDs in Use Register\_MS - Format

Offset	0xBASE_0DBC (JRJIDU_LS)																Access: Read-Only
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-70. Job Ring Job IDs in Use Register\_LS - Format****Table B-69. Job Ring Job IDs in Use Register - Field Descriptions**

Field	Description
<b>Most-significant portion of register</b>	
15 JID15	Job ID 15. Job ID 15 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
14 JID14	Job ID 14. Job ID 14 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
13 JID13	Job ID 13. Job ID 13 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
12 JID12	Job ID 12. Job ID 12 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
11 JID11	Job ID 11. Job ID 11 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
10 JID10	Job ID 10. Job ID 10 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
9 JID09	Job ID 09. Job ID 09 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
8 JID08	Job ID 08. Job ID 08 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
7 JID07	Job ID 07. Job ID 07 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
6 JID06	Job ID 06. Job ID 06 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
5 JID05	Job ID 05. Job ID 05 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
4 JID04	Job ID 04. Job ID 04 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
3 JID03	Job ID 03. Job ID 03 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.

## CAAM Register Descriptions

Field	Description
2 JID02	Job ID 02. Job ID 02 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
1 JID01	Job ID 01. Job ID 01 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
0 JID00	Job ID 00. Job ID 00 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
31-16	Reserved.
15 JID15	Job ID 15. Job ID 15 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
14 JID14	Job ID 14. Job ID 14 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
13 JID13	Job ID 13. Job ID 13 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
12 JID12	Job ID 12. Job ID 12 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
11 JID11	Job ID 11. Job ID 11 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
10 JID10	Job ID 10. Job ID 10 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
9 JID09	Job ID 09. Job ID 09 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
8 JID08	Job ID 08. Job ID 08 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
7 JID07	Job ID 07. Job ID 07 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
6 JID06	Job ID 06. Job ID 06 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
5 JID05	Job ID 05. Job ID 05 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
4 JID04	Job ID 04. Job ID 04 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
3 JID03	Job ID 03. Job ID 03 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
2 JID02	Job ID 02. Job ID 02 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
1 JID01	Job ID 01. Job ID 01 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
0 JID00	Job ID 00. Job ID 00 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
<b>Least-significant portion of register</b>	

Field	Description
31 BC	Been Changed. The BC bit is used to verify that consistent data has been read from the Address Array Registers. BC is set to 0 when JRAAA0_MS is read, and BC is then set to 1 if the content of any of the JRAAAx registers or the JRAAVS register changes before the JRJIDU_LS is read. So if BC is 1 after this sequence of register reads, some of the data that was read may be inconsistent with other data that was read. In this case the address Array registers should be read again.
30-0	Reserved.

### B.2.3.8 Job Ring Job-Done Job ID FIFO (JRJDJIFx)

The Job Queue maintains an ordered list of Job IDs for the completed jobs whose completion status is waiting to be written to a Job Ring output ring. The Job Ring Job-Done Job ID FIFO registers report the Job IDs in this ordered list. Note that these Job IDs are not reset as job status is written to output rings, so the completion status for some Job IDs that appear in these registers may already have been written to output rings.

Offset	0xBASE_0DC0 (JRJDJIF0) - The oldest 4 entries that were placed in the JRJ-D JID FIFO	Access:
		Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0 0   JOB_ID_ENTRY0   0 0 0 0   JOB_ID_ENTRY1	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   JOB_ID_ENTRY2   0 0 0 0   JOB_ID_ENTRY3	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-71. Job Ring Job-Done Job ID FIFO 0 - Format

## CAAM Register Descriptions

Offset	0xBASE_0DC4 (JRJDJIF1) - The 4 entries that were placed in the FIFO after those in JRJ-D JID FIFO	Access: Read-Only
0		
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
	0 0 0 0   JOB_ID_ENTRY4   0 0 0 0   JOB_ID_ENTRY5	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
	0 0 0 0   JOB_ID_ENTRY4   0 0 0 0   JOB_ID_ENTRY5	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
Offset	0xBASE_0DC8 (JRJDJIF2) - The 4 entries that were placed in the FIFO after those in JRJ-D JID FIFO	Access: Read-Only
1		
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
	0 0 0 0   JOB_ID_ENTRY8   0 0 0 0   JOB_ID_ENTRY9	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
	0 0 0 0   JOB_ID_ENTRY 10   0 0 0 0   JOB_ID_ENTRY 11	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-73. Job Ring Job-Done Job ID FIFO 2 - Format

Offset	0xBASE_0DCC (JRJDJIF3)- The most recent 4 entries that were placed in the FIFO	Access: Read-Only
0		
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
	0 0 0 0   JOB_ID_ENTRY 12   0 0 0 0   JOB_ID_ENTRY13	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
	0 0 0 0   JOB_ID_ENTRY 14   0 0 0 0   JOB_ID_ENTRY15	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-74. Job Ring Job-Done Job ID FIFO 3 - Format

**Table B-70. Job Ring Job-Done Job ID FIFO - Field Descriptions**

<b>Field</b>	<b>Description</b>
JRJDJIF0	
31-28	Reserved.
27-24 JOB_ID_ENTRY0	Job ID entry 0. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This is the oldest entry (closest to the head of the FIFO).
23-20	Reserved.
19-16 JOB_ID_ENTRY1	Job ID entry 1. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY0 in the FIFO.
15-12	Reserved.
11-8 JOB_ID_ENTRY2	Job ID entry 2. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY1 in the FIFO.
7-4	Reserved.
3-0 JOB_ID_ENTRY3	Job ID entry 3. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY2 in the FIFO.
JRJDJIF1	
31-28	Reserved.
27-24 JOB_ID_ENTRY4	Job ID entry 4. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY3 in the FIFO
23-20	Reserved.
19-16 JOB_ID_ENTRY5	Job ID entry 5. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY4 in the FIFO.
15-12	Reserved.
11-8 JOB_ID_ENTRY6	Job ID entry 6. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY5 in the FIFO.
7-4	Reserved.
3-0 JOB_ID_ENTRY7	Job ID entry 7. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY6 in the FIFO.
JRJDJIF2	
31-28	Reserved.
27-24 JOB_ID_ENTRY8	Job ID entry 8. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. TThis entry follows JOB_ID_ENTRY7 in the FIFO.
23-20	Reserved.
19-16 JOB_ID_ENTRY9	Job ID entry 9. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY8 in the FIFO.
15-12	Reserved.
11-8 JOB_ID_ENTRY10	Job ID entry 10. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY9 in the FIFO.
7-4	Reserved.

Field	Description
3-0 JOB_ID_ENTRY11	Job ID entry 11. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY10 in the FIFO.
	JRJDJIF3
31-28	Reserved.
27-24 JOB_ID_ENTRY12	Job ID entry 12. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY11 in the FIFO.
23-20	Reserved.
19-16 JOB_ID_ENTRY13	Job ID entry 13. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY12 in the FIFO.
15-12	Reserved.
11-8 JOB_ID_ENTRY14	Job ID entry 14. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY13 in the FIFO.
7-4	Reserved.
3-0 JOB_ID_ENTRY15	Job ID entry 15. This field contains the Job ID of a job whose completion status is waiting to be written to an output ring. This entry follows JOB_ID_ENTRY14 in the FIFO.

### B.2.3.9 Job Ring Job-Done Source FIFO (JRJDSFx)

The Job Queue keeps track of the job source (Job Ring numbers) for each Job ID. The Job Ring Job-Done Source FIFO registers report the job sources for each Job ID. Each entry in this FIFO is matched to corresponding entries in the JRJDVF FIFO and JRDDAx registers.

Offset	0xBASE_0DE0 (JRJDSF0) - The oldest 8 entries that were placed in the FIFO	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 SRC0   0 0 SRC1   0 0 SRC2   0 0 SRC3	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 SRC4   0 0 SRC5   0 0 SRC6   0 0 SRC7	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-75. Job Ring Job-Done Source FIFO 0 - Format

Offset	0xBASE_0DE4 (JRJDSF1) - The 8 entries that were placed in the FIFO after those in FIFO 0																Access: Read-Only	
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
W	0	0	SRC8		0	0	SRC9		0	0	SRC10		0	0	SRC11			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
W	0	0	SRC12		0	0	SRC13		0	0	SRC14		0	0	SRC15			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-76. Job Ring Job-Done Source FIFO 1 - Format****Table B-71. Job Ring Job-Done Source FIFO - Field Descriptions**

Field	Description
JRJDSF0	
31-30	Reserved.
29-28 SRC0	Source 0. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This is the oldest entry (i.e. at the head of the FIFO).
27-26	Reserved.
25-24 SRC1	Source 1. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC0 in the FIFO.
23-22	Reserved.
21-20 SRC2	Source 2. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC1 in the FIFO.
19-18	Reserved.
17-16 SRC3	Source 3. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC2 in the FIFO..
15-14	Reserved.
13-12 SRC4	Source 4. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC3 in the FIFO.
11-10	Reserved.
9-8 SRC5	Source 5. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC4 in the FIFO.
7-6	Reserved.
5-4 SRC6	Source 6. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC5 in the FIFO.
3-2	Reserved.
1-0 SRC7	Source 7. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC6 in the FIFO.

Field	Description
JRJDSF1	
31-30	Reserved.
29-28 SRC8	Source 8. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC7 in the FIFO.
27-26	Reserved.
25-24 SRC9	Source 9. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC8 in the FIFO.
23-22	Reserved.
21-20 SRC10	Source 10. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC9 in the FIFO.
19-18	Reserved.
17-16 SRC11	Source 11. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC10 in the FIFO..
15-14	Reserved.
13-12 SRC12	Source 12. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC11 in the FIFO.
11-10	Reserved.
9-8 SRC13	Source 13. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC12 in the FIFO.
7-6	Reserved.
5-4 SRC14	Source 14. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC13 in the FIFO.
3-2	Reserved.
1-0 SRC15	Source 15. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. This entry follows SRC14 in the FIFO. This is the newest entry (i.e. at the tail of the FIFO).

### B.2.3.10 Job Ring Job-Done Valid FIFO (JRJDVF)

The Job Ring Job-Done Valid FIFO indicates which entries in the JRJDJI, JRJDS and JRJDDA FIFOs currently contain valid information. The register is intended to be used when debugging descriptor execution via a Job Ring.

Offset	0xBASE_0DF0 (JRJDVF)																Access: Read-Only			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
R	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15				
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-77. Job Ring Job-Done Valid FIFO - Format****Table B-72. Job Ring Job-Done Valid FIFO- Field Descriptions**

Field	Description
31 V0	Valid 0. When V0=1, JRJDJI FIFO Entry 0 contains a valid Job ID, and JRJDS FIFO Entry 0 contains a valid source Job Ring number.
30 V1	Valid 1. When V1=1, JRJDJI FIFO Entry 1 contains a valid Job ID, and JRJDS FIFO Entry 1 contains a valid source Job Ring number.
29 V2	Valid 2. When V2=1, JRJDJI FIFO Entry 2 contains a valid Job ID, and JRJDS FIFO Entry 2 contains a valid source Job Ring number.
28 V3	Valid 3. When V3=1, JRJDJI FIFO Entry 3 contains a valid Job ID, and JRJDS FIFO Entry 3 contains a valid source Job Ring number.
27 V4	Valid 4. When V4=1, JRJDJI FIFO Entry 4 contains a valid Job ID, and JRJDS FIFO Entry 4 contains a valid source Job Ring number.
26 V5	Valid 5. When V5=1, JRJDJI FIFO Entry 5 contains a valid Job ID, and JRJDS FIFO Entry 5 contains a valid source Job Ring number.
25 V6	Valid 6. When V6=1, JRJDJI FIFO Entry 6 contains a valid Job ID, and JRJDS FIFO Entry 6 contains a valid source Job Ring number.
24 V7	Valid 7. When V7=1, JRJDJI FIFO Entry 7 contains a valid Job ID, and JRJDS FIFO Entry 7 contains a valid source Job Ring number.
23 V8	Valid 8. When V8=1, JRJDJI FIFO Entry 8 contains a valid Job ID, and JRJDS FIFO Entry 8 contains a valid source Job Ring number.
22 V9	Valid 9. When V9=1, JRJDJI FIFO Entry 9 contains a valid Job ID, and JRJDS FIFO Entry 9 contains a valid source Job Ring number.
21 V10	Valid 10. When V10=1, JRJDJI FIFO Entry 10 contains a valid Job ID, and JRJDS FIFO Entry 10 contains a valid source Job Ring number.
20 V11	Valid 11. When V11=1, JRJDJI FIFO Entry 11 contains a valid Job ID, and JRJDS FIFO Entry 11 contains a valid source Job Ring number.
19 V12	Valid 12. When V12=1, JRJDJI FIFO Entry 12 contains a valid Job ID, and JRJDS FIFO Entry 12 contains a valid source Job Ring number.

**Table B-72. Job Ring Job-Done Valid FIFO- Field Descriptions**

Field	Description
18 V13	Valid 13. When V13=1, JRJDJI FIFO Entry 13 contains a valid Job ID, and JRJDS FIFO Entry 13 contains a valid source Job Ring number.
17 V14	Valid 14. When V14=1, JRJDJI FIFO Entry 14 contains a valid Job ID, and JRJDS FIFO Entry 14 contains a valid source Job Ring number.
16 V15	Valid 15. When V15=1, JRJDJI FIFO Entry 15 contains a valid Job ID, and JRJDS FIFO Entry 15 contains a valid source Job Ring number.
15-0	Reserved.

**B.2.3.11 Job Ring Job-Done Status Register (JRJDS)**

The Job Ring Job-Done Status Register indicates whether consistent data has been read from the JRJDJI, JRJDS, and JRJDDA FIFOs. The register is intended to be used when debugging descriptor execution via a Job Ring.

Offset	0xBASE_0DF4 (JRJDS)																Access: Read-Only
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	BC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-78. Job Ring Job-Done Status Register - Format****Table B-73. Job Ring Job-Done Status Register - Field Descriptions**

Field	Description
31 BC	Been Changed. The BC bit is set to 0 when Job Ring Job-Done Descriptor Address register 0 is read. If any of the data in the JRJDJI, JRJDS or JRJDDA FIFO or JRJDDA registers changes before the Job Ring Job-Done Status register is read, the BC bit is set to 1. If the BC bit is 1 when the Job Ring Job-Done Status register is read, this indicates that inconsistent information was read from the JRJDJI, JRJDS or JRJDDA FIFO or JRJDDA registers. The registers should be reread to obtain a consistent set of data.
30-0	Reserved.

## B.2.4 Job Ring Job-Done Descriptor Address Registers (JRJDDAx)

The JRJDDAx registers are used to store the address of a job descriptor when the job is sent to a holding tank. There is one 64-bit register for each Job ID. JRJDDA0 is for Job ID 0, JRJDDA1 is for Job ID 1, etc. Because these addresses are updated only when a new job starts in a holding tank, some addresses read from these registers may be for completed jobs that have already been written to an output ring. These registers are intended to be used when debugging descriptor execution via a Job Ring.

Offset 0xBASE_0E00 (JRJDDA<x>_MS), offset 8, range x = 0 .. 15																Access: Read-Only			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
W																			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
W																			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure 0-1. Job Ring Job-Done Descriptor Address Registers\_MS - Format

Offset 0xBASE_0E04 (JRJDDA<x>_LS), offset 8, range x = 0 .. 15																Access: Read-Only			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R					JD_ADDR_LS														
W																			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R					JD_ADDR_LS														
W																			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure 0-2. Job Ring Job-Done Descriptor Address Registers\_LS - Format

Table B-74. Job Ring Job-Done Descriptor Address Registers - Field Descriptions

Field	Description
Most-significant portion of register	
31-0	Reserved.

Field	Description
Least-significant portion of register	
31-0 JD_ADDR_LS	Job Descriptor Address.

## B.2.5 Fault and Version ID Registers

These registers are used for obtaining bus fault information, and determining the version numbers of various subcomponents within CAAM. These registers would typically be accessed by driver processes.

### B.2.5.1 CHA Revision Number Register (CRNR)

The CHA Revision Number register indicates the revision number of each CHA. The revisions are numbered independently for each version of a particular CHA (see Section B.2.5.10 on page B-550). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit registers. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces. The register format is shown in the figure and table below.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FA0, 0xBASE_1FA0, 0xBASE_2FA0, 0xBASE_8FA0 (CRNR_MS)	Access: Read-Only																																																
R	<table border="1"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td colspan="4">JRRN</td><td colspan="4">DECORN</td><td colspan="4">Reserved</td><td colspan="4">Reserved</td> </tr> <tr> <td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	JRRN				DECORN				Reserved				Reserved				0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																			
JRRN				DECORN				Reserved				Reserved																																						
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0																																			
W	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="4">Reserved</td><td colspan="4">Reserved</td><td colspan="4">Reserved</td><td colspan="4">Reserved</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reserved				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																			
Reserved				Reserved				Reserved				Reserved																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																			
PO Reset																																																		

Figure B-79. CHA Revision Number Register\_MS - Format

Offset There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FA4, 0xBASE_1FA4, 0xBASE_2FA4, 0xBASE_8FA4 (CRNR_LS)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 Reserved Reserved Reserved RNGRN	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 MDRN ARC4RN DESRN AESRN	
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 1	

**Figure B-80. CHA Revision Number Register \_LS - Format**

**Table B-75. CHA Revision Number Register - Field Descriptions**

Field	Description
Most-significant half of CHA Revision Number Register	
31-28 JRRN	Job Ring Revision Number
27-24 DECO RN	DECO Revision Number
23-16	Reserved
15-12	Reserved
11-8	Reserved
7-4	Reserved
3-0	Reserved.
Least-significant half of CHA Revision Number Register	
31-28	Reserved.
27-24	Reserved.
23-20	Reserved.
19-16 RNGRN	Random Number Generator Revision Number.

**Table B-75. CHA Revision Number Register - Field Descriptions**

Field	Description
15-12 MDRN	Message Digest (Hashing) module Revision Number.
11-8 ARC4RN	ARC4 module Revision Number.
7-4 DESRN	DES module Revision Number.
3-0 AESRN	AES module Revision Number. 0000 - No Differential Power Analysis resistance implemented 0001 - Differential Power Analysis resistance is implemented

### B.2.5.2 Compile Time Parameters Register (CTPR)

The Compile Time Parameters register indicates the settings of the Verilog parameters at the time CAAM was compiled. Register contents are shown in [Figure B-81](#) and [Table B-76](#). Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit registers. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FA8, 0xBASE_1FA8, 0xBASE_2FA8, 0xBASE_8FA8 (CTPR_MS)																Access: Read-Only
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	AXI_PIPE_DEPTH				AXI_MID	AXI_PRI	QI	ACC_CTL	C1C2	0	PC	DECO_WD	PM_EVT_BUS	SG8	MCFG_PS	MCFG_BURST	
W																	
PO Reset	0	1	0	0	0	0	0	1	1	0	0	1	0	0	0	1	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	IP_CLK	0	0	0	RNG_I			0	0	0	0	0	0	0	0	
W																	
PO Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-81. Compile Time Parameters Register\_MS - Format**

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FAC, 0xBASE_1FAC, 0xBASE_2FAC, 0xBASE_8FAC (CTPR_LS)																Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16																
W																	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0																
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0																
W																	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 1 0																

**Figure B-82. Compile Time Parameters Register\_LS - Format**

**Table B-76. Compile Time Parameters Register - Field Descriptions**

Field	Description
Most-significant half of register	
31-28 AXI_PIPE_DEPTH	AXI Pipeline Depth. A value of 0b0000 indicates no limit.
27 AXI_MID	1 = The logic to select MIDs is included in CAAM
26 AXI_PRI	1 = The logic for the AXI Master Priority signals is included in CAAM
25 QI	1 = The Queue Manager Interface is included in CAAM
24 ACC_CTL	1 = CAAM implements MID-based access control for the IP Bus registers
23 C1C2	1 = CAAM implements Class 2 Key and Context registers that are separate from the Class 1 Key and Context registers
22	Reserved.
21 PC	1 = CAAM implements Performance Counter registers
20 DECO_WD	1 = CAAM implements a DECO Watchdog Counter
19 PM_EVT_BUS	Performance Monitor Event Bus. 1 = CAAM implements a Performance Monitor Event Bus.

## CAAM Register Descriptions

Field	Description
18 SG8	1 = CAAM implements eight Scatter-Gather Tables, rather than just one.
17 MCFG_PS	1 = The Master Configuration Register contains a Pointer Size field
16 MCFG_BURST	1 = The Master Configuration Register contains an AXI Burst field.
15	Reserved.
14 IP_CLK	IP Bus Slave Clock 1 = The frequency of CAAM's IP Bus Slave Clock is one-half the frequency of CAAM's AXI bus clock. 0 = The frequency of CAAM's IP Bus Slave Clock is the same as the frequency of CAAM's AXI bus clock..
13-11	Reserved.
10-8 RNG_I	RNG Instantiations. This indicates the number of RNG instantiations that are implemented. Note that each instantiation is the data context for an independent RNG stream, and may share the same RNG hardware as other instantiations. The number of hardware RNGs is indicated in the RNGNUM field of the CHANUM register.
Least-significant half of register	
31-15	Reserved.
14 SPLIT_KEY	1 = CAAM implements the split-key protocol.
13	Reserved.
12 DBL_CRC	1 = CAAM implements specialized support for 3G Double CRC.
11 P3G_LTE	1 = CAAM implements specialized support for 3G and LTE protocols.
10 RSA	1 = CAAM implements specialized support for RSA encrypt and decrypt operations.
9 MACSEC	1 = CAAM implements specialized support for the MACSEC protocol.
8 TLS_PRF	1 = CAAM implements specialized support for the TLS protocol pseudo-random function.
7 SSL_TLS	1 = CAAM implements specialized support for the SSL and TLS protocols.
6 IKE	1 = CAAM implements specialized support for the IKE protocol.
5 IPSEC	1 = CAAM implements specialized support for the IPSEC protocol.
4 SRTP	1 = CAAM implements specialized support for the SRTP protocol.
3 WIMAX	1 = CAAM implements specialized support for the WIMAX protocol.

Field	Description
2 WIFI	1 = CAAM implements specialized support for the WIFI protocol.
1 BLOB	1 = CAAM implements specialized support for encapsulating and decapsulating cryptographic blobs.
0 KG_DS	1 = CAAM implements specialized support for Public Key Generation and Digital Signatures.

### B.2.5.3 Secure Memory Status Register (SMSTA)

This register indicates the current status of the Secure Memory Controller. The status of the first error that occurs will remain until this register is read. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset 0xBASE\_0FB4, 0xBASE\_1FB4, 0xBASE\_2FB4, 0xBASE\_8FB4  
(SMSTA) Access: Read-Only

	31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16
R W	PART				0	PAGE													
PO Reset	0	0	0	0		0	0	0	0		0	0	0	0		0	0	0	0
R W	15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0
	0	0	0	0		MID					ACCERR					STATE			
PO Reset	0	0	0	0		0	0	0	0		0	0	0	0		0	0	0	0

**Figure B-83. Secure Memory Status Register - Format**

**Table B-77. Secure Memory Status Register - Field Descriptions**

Field	Description
31-28 PART	Partition. The number of the Secure Memory partition associated with the denied access. This field will be zero for ACCERR values 0h and 1h.
27	Reserved.
26-16 PAGE	Page. The number of pages that have completed zeroization during POR or following a transition to Fail Mode. The Secure Memory Controller updates this field as it zeroizes each page. Following completion of zeroization, this value remains constant until a Secure Memory access error occurs. When ACCERR != 0h, PAGE indicates the Secure Memory page associated with the denied access.
15-14	Reserved.

Field	Description																										
13-8 MID	The MID of the bus master whose access to Secure Memory was denied.																										
7-4 ACCERR	Access Error. This field indicates the reason that a read or write access to Secure Memory was denied. If an access was denied for more than one reason, the lowest numbered error value will be reported. <table border="1" data-bbox="274 397 1475 1241"> <thead> <tr> <th>Value</th><th>Access Error Description</th></tr> </thead> <tbody> <tr> <td>0h</td><td>No error occurred</td></tr> <tr> <td>1h</td><td>A bus transaction attempted to access a page in Secure Memory, but the page was not allocated to any partition.</td></tr> <tr> <td>2h</td><td>A bus transaction attempted to access a partition, but the transaction's MID was not granted access to the partition in the partition's SM2/1JR registers.</td></tr> <tr> <td>3h</td><td>A bus transaction attempted to write, but writes to this partition are not allowed.</td></tr> <tr> <td>4h</td><td>A bus transaction attempted to read, but reads from this partition are not allowed.</td></tr> <tr> <td>6h</td><td>A bus transaction attempted a non-key read, but the only reads permitted from this partition are key reads.</td></tr> <tr> <td>9h</td><td>Secure Memory Blob import or export was attempted, but Secure Memory Blob access is not allowed for this partition.</td></tr> <tr> <td>Ah</td><td>A Descriptor attempted a Secure Memory Blob import or export, but not all of the pages referenced were from the same partition.</td></tr> <tr> <td>Bh</td><td>A memory access was directed to Secure Memory, but the specified address is not implemented in Secure Memory. The address was either outside the address range occupied by Secure Memory, or was within an unimplemented portion of the 4k-byte address block occupied by a 1k-byte or 2k-byte Secure Memory page.</td></tr> <tr> <td>Ch</td><td>A bus transaction was attempted, but the burst would have crossed a page boundary.</td></tr> <tr> <td>Dh</td><td>An attempt was made to access a page while it was still being initialized.</td></tr> <tr> <td>All other values</td><td>are reserved.</td></tr> </tbody> </table>	Value	Access Error Description	0h	No error occurred	1h	A bus transaction attempted to access a page in Secure Memory, but the page was not allocated to any partition.	2h	A bus transaction attempted to access a partition, but the transaction's MID was not granted access to the partition in the partition's SM2/1JR registers.	3h	A bus transaction attempted to write, but writes to this partition are not allowed.	4h	A bus transaction attempted to read, but reads from this partition are not allowed.	6h	A bus transaction attempted a non-key read, but the only reads permitted from this partition are key reads.	9h	Secure Memory Blob import or export was attempted, but Secure Memory Blob access is not allowed for this partition.	Ah	A Descriptor attempted a Secure Memory Blob import or export, but not all of the pages referenced were from the same partition.	Bh	A memory access was directed to Secure Memory, but the specified address is not implemented in Secure Memory. The address was either outside the address range occupied by Secure Memory, or was within an unimplemented portion of the 4k-byte address block occupied by a 1k-byte or 2k-byte Secure Memory page.	Ch	A bus transaction was attempted, but the burst would have crossed a page boundary.	Dh	An attempt was made to access a page while it was still being initialized.	All other values	are reserved.
Value	Access Error Description																										
0h	No error occurred																										
1h	A bus transaction attempted to access a page in Secure Memory, but the page was not allocated to any partition.																										
2h	A bus transaction attempted to access a partition, but the transaction's MID was not granted access to the partition in the partition's SM2/1JR registers.																										
3h	A bus transaction attempted to write, but writes to this partition are not allowed.																										
4h	A bus transaction attempted to read, but reads from this partition are not allowed.																										
6h	A bus transaction attempted a non-key read, but the only reads permitted from this partition are key reads.																										
9h	Secure Memory Blob import or export was attempted, but Secure Memory Blob access is not allowed for this partition.																										
Ah	A Descriptor attempted a Secure Memory Blob import or export, but not all of the pages referenced were from the same partition.																										
Bh	A memory access was directed to Secure Memory, but the specified address is not implemented in Secure Memory. The address was either outside the address range occupied by Secure Memory, or was within an unimplemented portion of the 4k-byte address block occupied by a 1k-byte or 2k-byte Secure Memory page.																										
Ch	A bus transaction was attempted, but the burst would have crossed a page boundary.																										
Dh	An attempt was made to access a page while it was still being initialized.																										
All other values	are reserved.																										
3-0 STATE	Current State. This field represents the current state of the Secure Memory Controller. <table border="1" data-bbox="274 1305 659 1613"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0h</td><td>Reset State</td></tr> <tr> <td>1h</td><td>Initialize State</td></tr> <tr> <td>2h</td><td>Normal State</td></tr> <tr> <td>3h</td><td>Fail State.</td></tr> <tr> <td>Others</td><td>Reserved.</td></tr> </tbody> </table>	Value	Description	0h	Reset State	1h	Initialize State	2h	Normal State	3h	Fail State.	Others	Reserved.														
Value	Description																										
0h	Reset State																										
1h	Initialize State																										
2h	Normal State																										
3h	Fail State.																										
Others	Reserved.																										

### B.2.5.4 Secure Memory Partition Owner Register (SMPO)

When a Job Ring owner reads this partition from the address alias associated with the Job Ring, the value returned will indicate whether that Job Ring TZ/owns the partition, another

Job RingTZ/owns the partition, the partition is unowned, or the partition is unimplemented. Each allocated partition is allocated to one of CAAM's Job Rings, and the owner of that Job Ring is considered the owner of all partitions allocated to that Job Ring. Note that the SMPO is accessible via multiple addresses (one per Job Ring). If Job Ring x's owner (as determined by the JRxMID register) reads the SMPO register via the address associated with Job Ring x, then the partition owner field will return a 11 code for each partition owned by the MIDTZ/ value associated with Job Ring x, a 10 code for each partition owned by some other Job RingTZ/, and a 00 code for each partition that is currently unallocated. Note that the same entity could own more than one Job Ring, and can determine which partitions are owned by which Job Ring by reading the SMPO register from different addresses. If a master other than Job Ring x's owner reads the SMPO register via the address associated with Job Ring x, all 0s will be returned.

Offset or Trustzone SecureWorld0xBASE\_1FBC (SMPO\_0)(used by Job Ring 0)  
0xBASE\_2FBC (SMPO\_1)(used by Job Ring 1)

Access:  
Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PO15		PO14		PO13		PO12		PO11		PO10		PO9		PO8	
W																
PO Reset	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PO7		PO6		PO5		PO4		PO3		PO2		PO1		PO0	
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1 <sup>1</sup>

**Figure B-84. Secure Memory Partition Owner Register - Format**

<sup>1</sup> Note that immediately after POR PO0 will return 11b if read with ns=1, but will return 10b if read with ns=0.

**Table B-78. Secure Memory Partition Owner Register - Field Descriptions**

Field	Description											
31-30 PO15 29-28 PO14 27-26 PO13 25-24 PO12 23-22 PO11 21-20 PO10 19-18 PO9 17-16 PO8 15-14 PO7 13-12 PO6 11-10 PO5 9-8 PO4 7-6 PO3 5-4 PO2 3-2 PO1 1-0 PO0	<p>Partition Owner: When read by a Job Ring owner, these fields indicate if the Partition is owned by that Job Ring, another Job Ring, Unowned, or Unimplemented.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th><th style="text-align: center;">Description</th></tr> </thead> <tbody> <tr> <td style="text-align: center;">00</td><td>Available; Unowned. A Job Ring owner may claim this partition by writing to the appropriate SMJR register address alias. Note that the entire register will return all 0s if read by a entity that does not own the Job Ring associated with the SMPO address alias that was read.</td></tr> <tr> <td style="text-align: center;">01</td><td>Partition does not exist in this version</td></tr> <tr> <td style="text-align: center;">10</td><td>Another entity owns the partition. This partition is unavailable to the reader. If the reader attempts to de-allocate the partition or write to the SMJR register or SMJR register for this partition or allocate a page to or de-allocate a page from this partition the command will be ignored. (Note that if a CSP partition is de-allocated, all entities (including the owner that de-allocated the partition) will see a "10" value for that partition until all its pages have been zeroized.)</td></tr> <tr> <td style="text-align: center;">11</td><td>The entity that read the SMPO register owns the partition. Ownership is claimed when the the access permissions register (SMJR) of an available partition is first written.</td></tr> </tbody> </table>		Value	Description	00	Available; Unowned. A Job Ring owner may claim this partition by writing to the appropriate SMJR register address alias. Note that the entire register will return all 0s if read by a entity that does not own the Job Ring associated with the SMPO address alias that was read.	01	Partition does not exist in this version	10	Another entity owns the partition. This partition is unavailable to the reader. If the reader attempts to de-allocate the partition or write to the SMJR register or SMJR register for this partition or allocate a page to or de-allocate a page from this partition the command will be ignored. (Note that if a CSP partition is de-allocated, all entities (including the owner that de-allocated the partition) will see a "10" value for that partition until all its pages have been zeroized.)	11	The entity that read the SMPO register owns the partition. Ownership is claimed when the the access permissions register (SMJR) of an available partition is first written.
Value	Description											
00	Available; Unowned. A Job Ring owner may claim this partition by writing to the appropriate SMJR register address alias. Note that the entire register will return all 0s if read by a entity that does not own the Job Ring associated with the SMPO address alias that was read.											
01	Partition does not exist in this version											
10	Another entity owns the partition. This partition is unavailable to the reader. If the reader attempts to de-allocate the partition or write to the SMJR register or SMJR register for this partition or allocate a page to or de-allocate a page from this partition the command will be ignored. (Note that if a CSP partition is de-allocated, all entities (including the owner that de-allocated the partition) will see a "10" value for that partition until all its pages have been zeroized.)											
11	The entity that read the SMPO register owns the partition. Ownership is claimed when the the access permissions register (SMJR) of an available partition is first written.											

### B.2.5.5 Fault Address Register (FAR)

The Fault Address Register is used for software debugging of external memory access errors. This register will hold the value of the AXI address where a read or write error occurred. The read error address is aligned to the data bus address boundary of the data sample where the error occurred. The write error address is the starting address of the transaction, aligned to the data bus address boundary. Additional details concerning the bus transaction appear in the FADR (see Section [B.2.5.6 on page B-544](#)). The Fault Address Register contents are shown in [Figure B-85](#) and [Table B-79](#). Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces. The values in the Fault Address Register and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers have been read.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset 0xBASE\_0FC4, 0xBASE\_1FC4, 0xBASE\_2FC4, 0xBASE\_8FC4  
(FAR)

Access:  
Read-Only

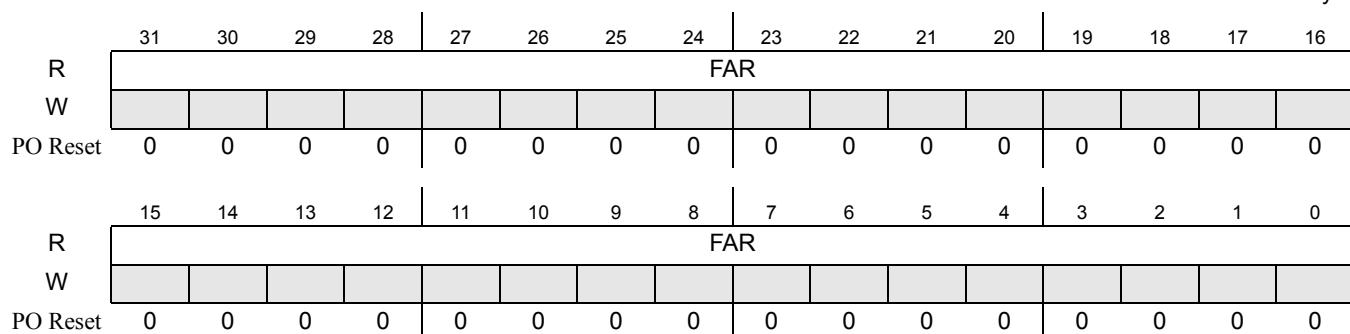


Figure B-85. Fault Address Register -Format

Table B-79. Fault Address Register - Field Descriptions

Field	Description
31-0 FAR	Fault Address. This is the AXI address at which the error occurred. If multiple errors occur, this is the AXI address at which the first error occurred. This address will remain in the register until software has read the register.

## B.2.5.6 Fault Address Detail Register (FADR)

The Fault Address Detail Register is used by software for debugging external memory access errors. This register will hold details about the AXI transaction where the error occurred. The associated AXI address is in the Fault Address Register (see Section [B.2.5.5 on page B-543](#)). The Fault Address Detail Register contents are shown in [Figure B-86](#) and [Table B-80](#). Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces. The values in the Fault Address Register and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers have been read.

## CAAM Register Descriptions

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FCC, 0xBASE_1FCC, 0xBASE_2FCC, 0xBASE_8FCC (FADR)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	FERR 0 FNS FBNDG FTDSC FKEY FKMOD	FSZ_EXT
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	0 0 0 0
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	DTYP JSRC BLKID TYP	FSZ
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	0 0 0 0

**Figure B-86. Fault Address Detail Register - Format**

**Table B-80. Fault Address Detail Register - Field Descriptions**

Field	Description
31-30 FERR	Fault Error Code. This is the AXI Error Response Code. 00 - OKAY - Normal Access 01 - Reserved 10 - SLVERR - Slave Error 11 - DECERR - Decode Error
29	Reserved. Always 0.
28	
27 FBNDG	Access permission binding access to Secure Memory. 0 - CAAM DMA was not reading access permissions from a Secure Memory partition at the time of the DMA error. 1 - CAAM DMA was reading access permissions from a Secure Memory partition at the time of the DMA error.
26 FTDSC	Trusted Descriptor access to Secure Memory 0 - CAAM DMA was not executing a Trusted Descriptor at the time of the DMA error. 1 - CAAM DMA was executing a Trusted Descriptor at the time of the DMA error.
25 FKEY	Key Access Read 0 - CAAM DMA was not attempting to perform a key read from Secure Memory at the time of the DMA error. 1 - CAAM DMA was attempting to perform a key read from Secure Memory at the time of the DMA error.
24 FKMOD	Key Modifier Read. (The SMJR register and SMAG1 JR and SMAG2 JR registers are used to modify the BEK during Secure Memory Blob export and import operations.) 0 - CAAM DMA was not attempting to read the key modifier from Secure Memory at the time that the DMA error occurred. 1 - CAAM DMA was attempting to read the key modifier from Secure Memory at the time that the DMA error occurred.
23-19	Reserved. Always 0.
18-16 FSZ_EXT	AXI Transaction Transfer Size - extended. This field holds the most significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error.
15 DTYP	Data Type. The type of data being processed when the AXI transfer error occurred. 0 - message data 1 - control data

Field	Description
14-12 JSRC	Job Source. The source of the job whose AXI transfer ended with an error: 000 Job Ring 0 001 Job Ring 1 010 reserved 011 reserved 100 reserved 101 reserved 110 reserved 111 reserved
11-8 BLKID	Block ID. The Block ID is the identifier of the block internal to CAAM that initiated the DMA transfer that resulted in an error. BLKID is interpreted as follows:  <u>BLKIDCAAM internal block</u> 4hJob Queue Controller Burst Buffer 5h One of the Job Rings (see JSRC field) 8hDECO0  All other values are reserved.
7 TYP	AXI Transaction Type. This is the type, read or write, of the DMA transaction that resulted in an error. 0 Read. 1 Write.
6-0 FSZ	AXI Transaction Transfer Size. This field holds the least-significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error. For large transfers the most-significant bits are held in field FSZ_EXT.

### B.2.5.7 CAAM Status Register (CSTA)

The CAAM Status Register indicates some status information that is relevant to the entire CAAM block. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces.

## CAAM Register Descriptions

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FD4, 0xBASE_1FD4, 0xBASE_2FD4, 0xBASE_8FD4 (CSTA)															Access: Read-Only
R																
W																
PO Reset																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	PLEND	MOO		0	0	0	0	0	TRNG_IDLE	IDLE	BSY
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure B-87. CAAM Status Register - Format

Table B-81. CAAM Status Register - Field Descriptions

Field	Description											
31-11	Reserved.											
10 PLEND	Platform Endianness. This is a hardwired SOC-specific configuration. PLEND indicates whether the CAAM bus master views memory <i>by default</i> as big endian or little endian. Software can override the default for particular data by setting bits in the Job Ring Configuration Register, and the DECO Job Queue Control Register. 0 - Platform default is Little Endian 1 - Platform default is Big Endian											
9-8 MOO	Mode of Operation. These bits indicate the Security Mode that CAAM is currently working in. The Security Mode is determined by the Security State Machine (see Security Monitor security states) located in the SNVS.  <table border="1"> <tr> <th>Value</th><th>Mode</th></tr> <tr> <td>00</td><td>Non-Secure</td></tr> <tr> <td>01</td><td>Secure</td></tr> <tr> <td>10</td><td>Trusted</td></tr> <tr> <td>11</td><td>Fail</td></tr> </table>		Value	Mode	00	Non-Secure	01	Secure	10	Trusted	11	Fail
Value	Mode											
00	Non-Secure											
01	Secure											
10	Trusted											
11	Fail											
7-3	Reserved.											
2 TRNG_IDLE	The TRNG portion of the RNG is idle. The free-running oscillator is stopped, so no entropy is being generated.											
1 IDLE	CAAM is idle. IDLE=1 indicates that CAAM is not currently processing any jobs from job rings and there are no pending interrupts (or the interrupts are masked) and the output job-ring timers are not counting. There may still be results in the Output Rings that have not been removed. Software should always check the Output Rings for results prior to checking for IDLE.											
0 BSY	CAAM Busy. This bit indicates that CAAM is processing at least one Descriptor.											

### B.2.5.8 Secure Memory Version ID Register (SMVID)

The Secure Memory Version ID register can be used by software to differentiate between different versions of the Secure Memory, and to determine the page size and the number of partitions and pages supported by this version of CAAM. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4k-byte address spaces. Since the register holds more than 32 bits, it is accessed as two 32-bit registers.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FD8, 0xBASE_1FD8, 0xBASE_2FD8, 0xBASE_8FD8																Access: Read-Only
<b>MAX_NPAG</b>																	
R	0	0	0	0	0	0											
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
<b>NPRT</b>																	
R	0	0											NPAG				
W																	
PO Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	

**Figure B-88. Secure Memory Version ID Register\_MS - Format**

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FDC, 0xBASE_1FDC, 0xBASE_2FDC, 0xBASE_8FDC																Access: Read-Only
<b>PSIZ</b>																	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
W																	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
<b>SMJV</b>																	
R													SMNV				
W																	
PO Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

**Figure B-89. Secure Memory Version ID Register\_LS - Format**

**Table B-82. Secure Memory Version ID Register - Field Descriptions**

Field	Description
Most-significant half of register	
31-26	Reserved.
25-16 MAX_ NPAG	Maximum allowable value for NPAG. This is the maximum number of Secure Memory pages that can be supported in this version of the CAAM hardware. The NPAG field indicates the actual number of Secure Memory pages that have been configured in this SOC. The actual number of pages may be smaller than the maximum that can be supported due to the size of the Secure Memory RAM that has been used in this SOC.
15-12 NPRT	This is the highest numbered Secure Memory partition, so there can be 1 to 16 partitions.
11-10	Reserved.
9-0 NPAG	This is the highest numbered page of Secure Memory. The number of the last page is in the range 0 to 1023, therefore the number of pages can range from 1 to 1024.
Least-significant half of register	
31-19	Reserved.
18-16 PSIZ	Page Size. The number of bytes in a Secure Memory page is $2^{\text{psiz}}$ KB. Zero means 1K-byte pages. The maximum page size is 128 KB.
15-8 SMJV	Secure Memory Major Version ID. A Zero value means that there is no Secure Memory in this version of CAAM
7-0 SMNV	Secure Memory Minor Version ID.

### B.2.5.9 CCB Version ID Register (CCBVID)

The CHA Cluster Block Version ID register can be used by software to differentiate between different versions of the CCB. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4 KB address spaces.

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FE4, 0xBASE_1FE4, 0xBASE_2FE4, 0xBASE_8FE4 (CCBVID)	Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 CAAM_ERA	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 AMJV	AMNV
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-90. CCB Version ID Register - Format

Table B-83. CCB Version ID Register - Field Descriptions

Field	Description
31-16 CAAM_ERA	CAAM Era. 00h - This version of CAAM is based on Era 5 or earlier RTL. 06h - This version of CAAM is based on Era 6 RTL. All other values are reserved.
23-16	Reserved.
15-8 AMJV	Accelerator Major Revision Number. This value will be incremented every time there is a major architectural change to the CCB design. Incrementing this results in the AMNV being set back to 0.
7-0 AMNV	Accelerator Minor Revision Number. This value will be incremented every time an RTL change has been made to the CCB module.

### B.2.5.10 CHA Version ID Register (CHAVID)

The CHA Version ID register can be used, along with the CCB Version ID, by software to differentiate between different versions of the cryptographic hardware accelerators. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4 KB address spaces.

## CAAM Register Descriptions

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FE8, 0xBASE_1FE8, 0xBASE_2FE8, 0xBASE_8FE8 (CHAVID_MS)	Access: Read-Only		
R	JRVID	DECOVID	Reserved	Reserved
W				
PO Reset	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0
R	Reserved	Reserved	Reserved	Reserved
W				
PO Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

**Figure B-91. CHA Version ID Register\_MS - Format**

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FEC, 0xBASE_1FEC, 0xBASE_2FEC, 0xBASE_8FEC (CHAVID_LS)	Access: Read-Only		
R	Reserved	Reserved	Reserved	RNGVID
W				
PO Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
R	MDVID	ARC4VID	DESVID	AESVID
W				
PO Reset	0 0 0 0	0 0 0 1	0 0 1 1	0 0 1 1

**Figure B-92. CHA Version ID Register\_LS - Format**

**Table B-84. CHA Version ID Register - Field Descriptions**

Field	Description
Most-significant half of CHA Version ID Register	
31-28 JRVID	Job Ring Version ID
27-24 DECO VID	DECO Version ID
23-16	Reserved
15-12	Reserved

**Table B-84. CHA Version ID Register - Field Descriptions**

Field	Description
11-8	Reserved
7-4	Reserved
3-0	Reserved
Least-significant half of CHA Version ID Register	
31-28	Reserved
27-24	Reserved
23-20	Reserved
19-16 RNGVID	Random Number Generator Version ID. 0010 - RNGB 0100 - RNG4
15-12 MDVID	Hashing module Version ID. 0000 - low-power MDHA, with SHA-1, SHA-256, SHA 224, MD5 & HMAC 0001 - low-power MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-384, MD5 & HMAC 0010 - high-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-384, MD5, HMAC & SMAC
11-8 ARC4VID	ARC4 module Version ID 0000 - low-power ARC4 0001 - high-performance ARC4
7-4 DESVID	DES module Version ID.
3-0 AESVID	AES module Version ID 0100 - high-performance AESA, implementing ECB, CBC, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, CBCXCB, CTRXCBC, XTS, and GCM modes 0011 - low-power AESA, implementing ECB, CBC, CFB128, OFB, CTR, CCM, CMAC and XCBC-MAC modes

### B.2.5.11 CHA Number Register (CHANUM)

The CHA Number register can be used by software to determine how many copies of each type of cryptographic hardware accelerator are implemented in this version of CAAM. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4 KB address spaces.

## CAAM Register Descriptions

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FF0, 0xBASE_1FF0, 0xBASE_2FF0, 0xBASE_8FF0 (CHANUM_MS)	Access: Read-Only
<b>JRNUM</b>		
R	31 30 29 28	27 26 25 24
W		
PO Reset	0 0 1 0	0 0 0 1
<b>ZANUM</b>		
R	15 14 13 12	11 10 9 8
W		
PO Reset	0 0 0 0	0 0 0 0
<b>DECONUM</b>		
R		
W		
PO Reset		
<b>Reserved</b>		
R		
W		
PO Reset		
<b>Reserved</b>		
R		
W		
PO Reset		
<b>SNW9NUM</b>		
R		
W		
PO Reset		
<b>CRCNUM</b>		
R		
W		
PO Reset		
<b>PKNUM</b>		
R		
W		
PO Reset		
<b>KASNUM</b>		
R		
W		
PO Reset		
<b>SNW8NUM</b>		
R		
W		
PO Reset		
<b>RNGNUM</b>		
R		
W		
PO Reset		
<b>MNDNUM</b>		
R		
W		
PO Reset		
<b>ARC4NUM</b>		
R		
W		
PO Reset		
<b>DESENUM</b>		
R		
W		
PO Reset		
<b>AESNUM</b>		
R		
W		
PO Reset		

**Figure B-93. CHA Number Register\_MS - Format**

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset	0xBASE_0FF4, 0xBASE_1FF4, 0xBASE_2FF4, 0xBASE_8FF4 (CHANUM_LS)	Access: Read-Only
<b>JRNUM</b>		
R	31 30 29 28	27 26 25 24
W		
PO Reset	0 0 0 0	0 0 0 0
<b>ZANUM</b>		
R	15 14 13 12	11 10 9 8
W		
PO Reset	0 0 0 1	0 0 0 1
<b>DECO NUM</b>		
R		
W		
PO Reset		
<b>SNW8NUM</b>		
R		
W		
PO Reset		
<b>RNGNUM</b>		
R		
W		
PO Reset		
<b>PKNUM</b>		
R		
W		
PO Reset		
<b>KASNUM</b>		
R		
W		
PO Reset		
<b>SNW9NUM</b>		
R		
W		
PO Reset		
<b>MNDNUM</b>		
R		
W		
PO Reset		
<b>ARC4NUM</b>		
R		
W		
PO Reset		
<b>DESENUM</b>		
R		
W		
PO Reset		
<b>AESNUM</b>		
R		
W		
PO Reset		

**Figure B-94. CHA Number Register\_LS - Format**

**Table B-85. CHA Number Register - Field Descriptions**

Field	Description
Most-significant half of CHA Number Register	
31-28 <b>JRNUM</b>	The number of copies of the Job Ring that are implemented in this version of CAAM
27-24 <b>DECO NUM</b>	The number of copies of the DECO that are implemented in this version of CAAM
23-16	Reserved
15-12 <b>ZANUM</b>	The number of copies of ZUCA that are implemented in this version of CAAM

**Table B-85. CHA Number Register - Field Descriptions**

Field	Description
11-8 ZENUM	The number of copies of ZUCE that are implemented in this version of CAAM
7-4 SNW9NUM	The number of copies of the SNOW-f9 module that are implemented in this version of CAAM
3-0 CRCNUM	The number of copies of the CRC module that are implemented in this version of CAAM
Least-significant half of CHA Number Register	
31-28 PKNUM	The number of copies of the Public Key module that are implemented in this version of CAAM
27-24 KASNUM	The number of copies of the Kasumi module that are implemented in this version of CAAM
23-20 SNW8NUM	The number of copies of the SNOW-f8 module that are implemented in this version of CAAM
19-16 RNGNUM	The number of copies of the Random Number Generator that are implemented in this version of CAAM.
15-12 MDNUM	The number of copies of the MDHA (Hashing module) that are implemented in this version of CAAM.
11-8 ARC4NUM	The number of copies of the ARC4 module that are implemented in this version of CAAM.
7-4 DESENUM	The number of copies of the DES module that are implemented in this version of CAAM.
3-0 AESNUM	The number of copies of the AES module that are implemented in this version of CAAM.

### B.2.5.12 CAAM Version ID Register (CAAMVID)

This register contains the ID for CAAM and major and minor revision numbers. It also contains the integration options, ECO revision, and configuration options. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple processors, this register is aliased to addresses in multiple 4 KB address spaces. The register and its fields are described in the figure and table below.

## CAAM Register Descriptions

Offset 0xBASE\_0BF8, 0xBASE\_0FF8, 0xBASE\_1FF8, 0xBASE\_2FF8, 0xBASE\_8FF8  
 (CAAMVID\_MS)

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IP_ID															
W																
PO Reset	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	0
R	MAJ_REV															MIN_REV
W																
PO Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure B-95. CAAM Version ID Register\_MS - Format

There is only one copy of this register, but the register is accessible at all the addresses shown below:

Offset 0xBASE\_0BFC, 0xBASE\_0FFC, 0xBASE\_1FFC, 0xBASE\_2FFC, 0xBASE\_8FFC  
 (CAAMVID\_LS)

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	COMPILE_OPT								INTG_OPT							
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	ECO_REV								CONFIG_OPT							
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-96. CAAM Version ID Register\_LS - Format

Table B-86. CAAM Version ID Register - Field Descriptions

Field	Description
Most-significant half of CAAM Version ID Register	
31-16 IP_ID	ID for CAAM.
15-8 MAJ_REV	Major revision number for CAAM.
7-0 MIN_REV	Minor revision number for CAAM.
Least-significant half of CAAM Version ID Register	
31-24 COMPILE_OPT	Compile options for CAAM.
23-16 INTG_OPT	Integration options for CAAM.

**Table B-86. CAAM Version ID Register - Field Descriptions**

Field	Description
15-8 ECO_REV	ECO revision for CAAM.
7-0 CONFIG_OPT	Configuration options for CAAM.

## B.2.6 Job Ring Registers

This section describes registers found in each of the job rings. A functional description of the job rings can be found in the “Cryptographic Acceleration and Assurance Module (CAAM)” chapter.

If the job ring is allocated to TrustZone SecureWorld (JRSDIDR TZ=1), then the job ring registers associated with this ring may only be written via a bus transaction that has ns=0. Transactions with ns=1 for job rings owned by TrustZone SecureWorld will be ignored.

### B.2.6.1 Input Ring Base Address Register (IRBAR)

The Input Ring Base Address register holds the physical address of the Input Ring in memory. Because there are two Job Rings, there are two copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRBAR may only be written with a transaction with ns=0. The IRBAR register can be written only when there are no jobs in the Input Ring or when the Job Ring is halted, else an *Input Ring base address invalid write* error will result and a Job Ring reset or a power on reset will be required. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRBAR the new head of the queue within the Input Ring will be located at the value just written to the IRBAR. Note that if the Input Ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost. The address written to the Input Ring Base

Address register must be 4-byte aligned, else an error will result and the Job Ring will not process jobs until a valid address is written and the error is cleared.

Offset	0xBASE_1004 (IRBAR0_LS)(used by Job Ring 0)	Accessible only when using MID specified in JR0MIDR register	Access: Read/Write
	0xBASE_2004 (IRBAR1_LS)(used by Job Ring 1)	Accessible only when using MID specified in JR1MIDR register	
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16		
W	IRBA		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0		
W	IRBA		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		

**Figure B-97. Input Ring Base Address Register - Format**

**Table B-87. Input Ring Base Address Register - Field Descriptions**

Field	Description
31-0 IRBA	Input Ring Base Address.

### B.2.6.2 Input Ring Size Register (IRSR)

The Input Ring Size register holds the current size of the Input Ring, measured in number of entries. Since each entry consists of one address pointer, an entry is a single 32-bit word. Because there are 2 Job Rings, there are 2 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRSR may only be written with a transaction with ns=0. This register should be written only when there are no jobs in the ring, or the Job Ring is halted. No error is generated, but jobs may be lost. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRSR the new head of the queue within the Input Ring will be located at the value stored in the IRBAR. Note that if the Input Ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost.

The size of the pointer entries in the ring is one word.

Offset	0xBASE_100C (IRSR0)(used by Job Ring 0)	Access: Read/Write														
	0xBASE_200C (IRSR1)(used by Job Ring 1)	Accessible only when using MID specified in JR0MIDR register Accessible only when using MID specified in JR1MIDR register														
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	IRS						0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-98. Input Ring Size Register - Format****Table B-88. Input Ring Size Register - Field Descriptions**

Field	Description
31-10	Reserved. Must be 0.
9-0 IRS	Input Ring Size. (measured in number of entries)

### B.2.6.3 Input Ring Slots Available Register (IRSAR)

The Input Ring Slots Available Register gives the number of empty slots for jobs in the Input Ring. Since each slot consists of one address pointer, a job slot is a single 32-bit word. Because there are 2 Job Rings, there are 2 copies of this register. This tells software how many more jobs it can submit to CAAM before the Input Ring would be full. CAAM increments this register when it removes a job from the Input Ring for processing. CAAM decrements this register by the value in the Input Ring Jobs Added Register (see Section [B.2.6.4 on page B-559](#)) when that register is updated. The value of the Input Ring Slots Available Register will never be larger than the Input Ring Size Register (see Section [B.2.6.2 on page B-557](#)).

When the Job Ring is allocated to TrustZone SecureWorld, IRSAR may only be written with a transaction with ns=0.

Offset	0xBASE_1014 (IRSAR0)(used by Job Ring 0)	Accessible only when using MID specified in JR0MIDR register	Access: Read-Only
	0xBASE_2014 (IRSAR1)(used by Job Ring 1)		Accessible only when using MID specified in JR1MIDR register
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16		
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	IRSA	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0		

**Figure B-99. Input Ring Slots Available Register - Format****Table B-89. Input Ring Slots Available Register - Field Descriptions**

Field	Description
31-10	Reserved. Must be 0.
9-0 IRSA	Input Ring Slots Available. (measured in number of available job slots)

### B.2.6.4 Input Ring Jobs Added Register (IRJAR)

The Input Ring Jobs Added Register tells CAAM how many new jobs were added to the Input Ring. Because there are 2 Job Rings, there are 2 copies of this register. Software must write into this register the number of Job Descriptor addresses that software has added to the ring. When the Input Ring Jobs Added Register is written, CAAM adds that new value to its count of the jobs available for processing and decrements the Input Ring Slots Available Register. The value in the Input Ring Jobs Added Register must not be larger than the value of the Input Ring Slots Available Register (see Section [B.2.6.3 on page B-558](#)). If more jobs are added than the value in the Input Ring Slots Available Register an "Added too many jobs" error (type 9h) will occur. This is a fatal error and will require a Job Ring reset or power on reset to correct.

When the Job Ring is allocated to TrustZone SecureWorld, IRJAR may only be written with a transaction with ns=0.

Offset	0xBASE_101C (IRJAR0)(used by Job Ring 0)	Accessible only when using MID specified in JR0MIDR register	Access: Read/Write														
	0xBASE_201C (IRJAR1)(used by Job Ring 1)		Accessible only when using MID specified in JR1MIDR register														
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	0	0	0	0	0	0	IRJA							0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-100. Input Ring Jobs Added Register - Format

Table B-90. Input Ring Jobs Added Register - Field Descriptions

Field	Description
31-10	Reserved. Must be 0.
9-0 IRJA	Input Ring Jobs Added. (measured in number of entries)

### B.2.6.5 Output Ring Base Address Register (ORBAR)

The Output Ring Base Address Register holds the address of the Output Ring in memory. Because there are 2 Job Rings, there are 2 copies of this register. When the Job Ring is allocated to TrustZone SecureWorld, ORBAR may only be written with a transaction with ns=0. This register can be written only when the Job Ring is halted or when there are no jobs from this ring in progress within CAAM or in the Input Ring or Output Ring, else an *Output Ring base address invalid write* error will result and a Job Ring reset, software CAAM reset or a power on reset will be required. Writing this register resets the Output Ring Write Index register, therefore following a write to the ORBAR the new tail of the queue within the Output Ring will be located at the value just written to the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that Job Ring will either still be in the Input Ring or will be completed and written to the Output Ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the Input Ring before writing the new Output Ring base address. This would allow for a clean start with a new empty Output Ring. Note that if the Output Ring was not empty at the time the ORBAR was written, those old results entries will not be in the new

Output Ring. The address written to the Output Ring Base Address register must be 4-byte aligned, else an error will result and the Job Ring will not process jobs until a valid address is written and the error is cleared.

Offset	0xBASE_1024 (ORBAR0)(used by Job Ring 0)	Access: Read/Write
	0xBASE_2024 (ORBAR1) (used by Job Ring 1)	Accessible only when using MID specified in JR0MIDR register Accessible only when using MID specified in JR1MIDR register
PO Reset	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	ORBA
	R W	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0
	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	ORBA
PO Reset	R W	
	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0

**Figure B-101. Output Ring Base Address Register - Format**

**Table B-91. Output Ring Base Address Register - Field Descriptions**

Field	Description
31-0 ORBA	Output Ring Base Address.

### B.2.6.6 Output Ring Size Register (ORSR)

The Output Ring Size Register holds the current size of the Output Ring, measured in number of entries. Each entry in the Output Ring consists of one descriptor address pointer plus one 32-bit results status word, plus an optional word indicating the length of the SEQ sequence, if any, associated with this job (see INCL\_SEQ\_OUT field in the Job Ring Configuration Register, Section [B.2.6.11](#) on page [B-573](#)).

Because there are 2 Job Rings, there are 2 copies of this register. When the Job Ring is allocated to TrustZone SecureWorld, ORSR may only be written with a transaction with ns=0. This register should be written only when the Job Ring is halted or the Input Ring is empty and no jobs from this ring are in progress within CAAM. No error will be generated, but failure to observe these restrictions may lead to unpredictable results.

Writing this register resets the Output Ring Write Index register, therefore following a write to the ORSR the new tail of the queue within the Output Ring will be located at the value stored in the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that

Job Ring will either still be in the Input Ring or will be completed and written to the Output Ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the Input Ring before writing the new Output Ring base address. This would allow for a clean start with a new empty Output Ring. Note that if the Output Ring was not empty at the time the ORSR was written, those old results entries will not be in the new Output Ring. If the Output Ring is not empty when the ORSR is written, software may need to process or relocate those entries to avoid losing job results.

Offset	Access: Read/Write																
	0xBASE_102C (ORSR0) (used by Job Ring 0)				Accessible only when using MID specified in JR0MIDR register												
0xBASE_202C (ORSR1) (used by Job Ring 1)				Accessible only when using MID specified in JR1MIDR register													
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	0	0	0	0	0	0	ORS										
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-102. Output Ring Size Register - Format**

**Table B-92. Output Ring Size Register - Field Descriptions**

Field	Description
31-10	Reserved. Must be 0.
9-0 ORS	Output Ring Size. (measured in number of entries)

### B.2.6.7 Output Ring Jobs Removed Register (ORJRR)

The Output Ring Jobs Removed Register tells CAAM how many jobs were removed from the Output Ring for processing by software. Because there are 2 Job Rings, there are 2 copies of this register. Software must write into this register the number of entries that software has removed from the ring. When the Output Ring Jobs Removed Register is written, CAAM will subtract this amount from the Output Ring Slots Full Register (see Section [B.2.6.8 on page B-563](#)). The value of the Output Ring Jobs Removed Register must not be larger than the value in the Output Ring Slots Full Register. If a value larger than the

Output Ring Slots Full Register is written to the ORJRR, a *removed too many jobs error* will occur and a Job Ring reset, software CAAM reset or a power on reset will be required.

When the Job Ring is allocated to TrustZone SecureWorld, ORJRR may only be written with a transaction with ns=0.

Offset	Access: Read/Write															
	0xBASE_1034 (ORJRR0) (used by Job Ring 0)				Accessible only when using MID specified in JR0MIDR register				0xBASE_2034 (ORJRR1) (used by Job Ring 1)				Accessible only when using MID specified in JR1MIDR register			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	ORJR				0	0	0	0	0	0
PO Reset	0	0	0	0	0	0					0	0	0	0	0	0

Figure B-103. Output Ring Jobs Removed Register - Format

Table B-93. Output Ring Jobs Removed Register - Field Descriptions

Field	Description
31-10	Reserved. Must be 0.
9-0 ORJR	Output Ring Jobs Removed. (measured in number of entries)

### B.2.6.8 Output Ring Slots Full Register (ORSFR)

The Output Ring Slots Full Register tells the software how many completed jobs CAAM has placed in the Output Ring. Because there are 2 Job Rings, there are 2 copies of this register. CAAM will increment this register as it completes a Descriptor and adds it to the Output Ring. CAAM will decrement this register when software writes a new value to the Output Ring Jobs Removed Register (see Section [B.2.6.7 on page B-562](#)). The value in the Output Ring Slots Full Register cannot be larger than the value in the Output Ring Size Register (see Section [B.2.6.6 on page B-561](#)).

When the Job Ring is allocated to TrustZone SecureWorld, ORSFR may only be written with a transaction with ns=0.

Offset	0xBASE_103C (ORSFR0) (used by Job Ring 0)	Access: Read-Only														
	0xBASE_203C (ORSFR1) (used by Job Ring 1)	Accessible only when using MID specified in JR0MIDR register														
	0xBASE_303C (ORSFR2) (used by Job Ring 2)	Accessible only when using MID specified in JR1MIDR register														
	0xBASE_403C (ORSFR3) (us)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-104. Output Ring Slots Full Register - Format**

**Table B-94. Output Ring Slots Full Register - Field Descriptions**

Field	Description
31-10	Reserved. Must be 0.
9-0 ORSF	Output Ring Slots Full. (measured in number of entries)

### B.2.6.9 Job Ring Output Status Register (JRSTAR)

This register is used to show the status of the last job that was completed. Because there are 2 Job Rings, there are 2 copies of this register. Although it is possible to read the job completion status directly from this register, in normal circumstances this is not useful because the status value will quickly be overwritten when the next job completes. Bits 31-0 of this register are written into the Output Ring after the completion of a job, and software should read the status from there. Only one type of error will be valid at a time. The format of the status error details are different depending on the status source specified in the SSRC field in the Job Ring Output Status Register. This will be described in detail in the following subsections.

Offset	0xBASE_1044 (JRSTAR0) (used by Job Ring 0)	Accessible only when using MID specified in JR0MIDR register
	0xBASE_2044 (JRSTAR1) (used by Job Ring 1)	Accessible only when using MID specified in JR1MIDR register
		Access: Read-Only
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
	SSRC	SSED
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
R		SSED
W		
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-105. Job Ring Output Status Register - Format****Table B-95. Job Ring Output Status Register - Field Descriptions**

Field	Description																	
31-28 SSRC	Status source. These bits define which source is reporting the status.																	
	<table border="1"> <thead> <tr> <th>SSRC</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>No Status Source (No Error or Status Reported) See Section <a href="#">B.2.6.9.1 on page B-565</a> for SSED format.</td> </tr> <tr> <td>0010</td> <td>CCB Status Source (CCB Error Reported) See Section <a href="#">B.2.6.9.2 on page B-566</a> for SSED format.</td> </tr> <tr> <td>0011</td> <td>Jump Halt User Status Source (User-Provided Status Reported) See Section <a href="#">B.2.6.9.3 on page B-568</a> for SSED format.</td> </tr> <tr> <td>0100</td> <td>DECO Status Source (DECO Error Reported) See Section <a href="#">B.2.6.9.4 on page B-568</a> for SSED format.</td> </tr> <tr> <td>0110</td> <td>Job Ring Status Source (Job Ring Error Reported) See Section <a href="#">B.2.6.9.5 on page B-570</a> for SSED format.</td> </tr> <tr> <td>0111</td> <td>Jump Halt Condition Codes (Condition Code Status Reported) See Section <a href="#">B.2.6.9.6 on page B-571</a> for SSED format.</td> </tr> <tr> <td colspan="2">All other values are reserved.</td></tr> </tbody> </table>		SSRC	Details	0000	No Status Source (No Error or Status Reported) See Section <a href="#">B.2.6.9.1 on page B-565</a> for SSED format.	0010	CCB Status Source (CCB Error Reported) See Section <a href="#">B.2.6.9.2 on page B-566</a> for SSED format.	0011	Jump Halt User Status Source (User-Provided Status Reported) See Section <a href="#">B.2.6.9.3 on page B-568</a> for SSED format.	0100	DECO Status Source (DECO Error Reported) See Section <a href="#">B.2.6.9.4 on page B-568</a> for SSED format.	0110	Job Ring Status Source (Job Ring Error Reported) See Section <a href="#">B.2.6.9.5 on page B-570</a> for SSED format.	0111	Jump Halt Condition Codes (Condition Code Status Reported) See Section <a href="#">B.2.6.9.6 on page B-571</a> for SSED format.	All other values are reserved.	
SSRC	Details																	
0000	No Status Source (No Error or Status Reported) See Section <a href="#">B.2.6.9.1 on page B-565</a> for SSED format.																	
0010	CCB Status Source (CCB Error Reported) See Section <a href="#">B.2.6.9.2 on page B-566</a> for SSED format.																	
0011	Jump Halt User Status Source (User-Provided Status Reported) See Section <a href="#">B.2.6.9.3 on page B-568</a> for SSED format.																	
0100	DECO Status Source (DECO Error Reported) See Section <a href="#">B.2.6.9.4 on page B-568</a> for SSED format.																	
0110	Job Ring Status Source (Job Ring Error Reported) See Section <a href="#">B.2.6.9.5 on page B-570</a> for SSED format.																	
0111	Jump Halt Condition Codes (Condition Code Status Reported) See Section <a href="#">B.2.6.9.6 on page B-571</a> for SSED format.																	
All other values are reserved.																		
27-0 SSED	Source specific error details. The allowed values in this field depend on the status source specified in the SSRC field. See SSRC field above for links to the sections where the source-specific formats of the SSED field are defined.																	

**B.2.6.9.1 JRSTA Register SSED Field When SSRC= 0000 (No Status Source)**

When the SSRC value indicates no error or status source (SSRC=0000), then bits 27-0 of the Job Ring Output Status Register will be 0.

27 26 25 24   23 22 21 20   19 18 17 16   15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	0000_0000h
---	------------

**Figure B-106. JRSTA Register SSED Field when SSRC= No Status Source - Format**

### B.2.6.9.2 JRSTA Register SSED Field When SSRC= 0010 (CCB Status Source)

When the SSRC value indicates the status source is CCB (SSRC=0010), then bits 27-0 are as shown in [Figure B-107](#).

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure B-107. JRSTA Register SSED Field when SSRC Field = CCB - Format**

**Table B-96. JRSTA Register SSED Field when SSRC Field = CCB - Field Descriptions**

Field	Description
27 JMP	Jump. The Descriptor made a jump to another Descriptor. When this bit is set, the DESC INDEX field will contain the index into the Descriptor that was jumped to, and not to the original Descriptor.
26-16	Reserved.
15-8 DESC INDEX	Index into the failing Descriptor. Index is number of words from start of Descriptor. In some cases this value may be off by one or more words due to timing issues.

**Table B-96. JRSTA Register SSED Field when SSRC Field = CCB - Field Descriptions**

Field	Description																																																																																																														
7-4 CHAID	<p>Algorithm that generated the error. These bits define which algorithm is associated with the code in ERRID.</p> <table border="1"> <thead> <tr> <th>CHAID</th><th>Description</th></tr> </thead> <tbody> <tr><td>0001</td><td>AES</td></tr> <tr><td>0010</td><td>DES, 3DES</td></tr> <tr><td>0011</td><td>ARC4</td></tr> <tr><td>0100</td><td>MD5, SHA-1, SH-224, SHA-256</td></tr> <tr><td>0101</td><td>RNG</td></tr> <tr><td colspan="2">All other values are reserved.</td></tr> </tbody> </table>					CHAID	Description	0001	AES	0010	DES, 3DES	0011	ARC4	0100	MD5, SHA-1, SH-224, SHA-256	0101	RNG	All other values are reserved.																																																																																													
CHAID	Description																																																																																																														
0001	AES																																																																																																														
0010	DES, 3DES																																																																																																														
0011	ARC4																																																																																																														
0100	MD5, SHA-1, SH-224, SHA-256																																																																																																														
0101	RNG																																																																																																														
All other values are reserved.																																																																																																															
3-0 ERRID	<p>Descriptor Error ID.</p> <table border="1"> <thead> <tr> <th>ERRID Code</th><th>Valid with Class 1</th><th>Valid with Class 2</th><th>Valid with RNG</th><th>Details</th></tr> </thead> <tbody> <tr><td>0h</td><td>X</td><td>X</td><td>X</td><td>None. No error.</td></tr> <tr><td>1h</td><td>X</td><td>X</td><td>X</td><td>Mode error.</td></tr> <tr><td>2h</td><td>X</td><td>X</td><td>X</td><td>Data size error.</td></tr> <tr><td>3h</td><td>X</td><td>X</td><td></td><td>Key size error.</td></tr> <tr><td>3h</td><td></td><td></td><td>X</td><td>Instantiate error.</td></tr> <tr><td>4h</td><td></td><td></td><td>X</td><td>Not instantiated error.</td></tr> <tr><td>5h</td><td></td><td></td><td>X</td><td>Test Instantiation error.</td></tr> <tr><td>6h</td><td></td><td></td><td>X</td><td>Prediction Resistance error.</td></tr> <tr><td>6h</td><td>X</td><td>X</td><td></td><td>Data arrived out of sequence error.</td></tr> <tr><td>7h</td><td></td><td></td><td>X</td><td>Prediction Resistance and Test Request error.</td></tr> <tr><td>8h</td><td></td><td></td><td>X</td><td>Uninstantiate error.</td></tr> <tr><td>9h</td><td></td><td></td><td>X</td><td>RNG Secure Key Generation error.</td></tr> <tr><td>9h</td><td>X</td><td></td><td></td><td>DES key parity error.</td></tr> <tr><td>Ah</td><td>X</td><td>X</td><td></td><td>ICV check failed.</td></tr> <tr><td>Bh</td><td>X</td><td>X</td><td>X</td><td>Hardware error.</td></tr> <tr><td>Ch</td><td>X</td><td></td><td></td><td>Unsupported CCM AAD size.</td></tr> <tr><td>Dh</td><td>X</td><td></td><td></td><td>Class 1 CHA is not reset.</td></tr> <tr><td>Eh</td><td>X</td><td>X</td><td></td><td>Invalid CHA combination was selected</td></tr> <tr><td>Fh</td><td>X</td><td>X</td><td></td><td>Invalid CHA selected.</td></tr> <tr><td colspan="2">All other values are reserved.</td><td></td><td></td><td></td><td></td></tr> </tbody> </table>					ERRID Code	Valid with Class 1	Valid with Class 2	Valid with RNG	Details	0h	X	X	X	None. No error.	1h	X	X	X	Mode error.	2h	X	X	X	Data size error.	3h	X	X		Key size error.	3h			X	Instantiate error.	4h			X	Not instantiated error.	5h			X	Test Instantiation error.	6h			X	Prediction Resistance error.	6h	X	X		Data arrived out of sequence error.	7h			X	Prediction Resistance and Test Request error.	8h			X	Uninstantiate error.	9h			X	RNG Secure Key Generation error.	9h	X			DES key parity error.	Ah	X	X		ICV check failed.	Bh	X	X	X	Hardware error.	Ch	X			Unsupported CCM AAD size.	Dh	X			Class 1 CHA is not reset.	Eh	X	X		Invalid CHA combination was selected	Fh	X	X		Invalid CHA selected.	All other values are reserved.					
ERRID Code	Valid with Class 1	Valid with Class 2	Valid with RNG	Details																																																																																																											
0h	X	X	X	None. No error.																																																																																																											
1h	X	X	X	Mode error.																																																																																																											
2h	X	X	X	Data size error.																																																																																																											
3h	X	X		Key size error.																																																																																																											
3h			X	Instantiate error.																																																																																																											
4h			X	Not instantiated error.																																																																																																											
5h			X	Test Instantiation error.																																																																																																											
6h			X	Prediction Resistance error.																																																																																																											
6h	X	X		Data arrived out of sequence error.																																																																																																											
7h			X	Prediction Resistance and Test Request error.																																																																																																											
8h			X	Uninstantiate error.																																																																																																											
9h			X	RNG Secure Key Generation error.																																																																																																											
9h	X			DES key parity error.																																																																																																											
Ah	X	X		ICV check failed.																																																																																																											
Bh	X	X	X	Hardware error.																																																																																																											
Ch	X			Unsupported CCM AAD size.																																																																																																											
Dh	X			Class 1 CHA is not reset.																																																																																																											
Eh	X	X		Invalid CHA combination was selected																																																																																																											
Fh	X	X		Invalid CHA selected.																																																																																																											
All other values are reserved.																																																																																																															

### B.2.6.9.3 JRSTA Register SSED Field When SSRC = 0011 (Jump Halt User Status Source)

When the SSRC value indicates the status source is Jump Halt User Status (SSRC=0011), then bits 27-0 are as shown in [Figure B-108](#).

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-108. JRSTA Register SSED Field when SSRC=Jump Halt User Status - Format**

**Table B-97. JRSTA Register SSED Field when SSRC=Jump Halt User Status - Field Descriptions**

Field	Description
27 JMP	Jump. The Descriptor made a jump to another Descriptor. When this bit is set, the DESC INDEX field will contain the index into the Descriptor that was jumped to, and not to the original Descriptor.
26-16	Reserved.
15-8 DESC INDEX	Index into the failing Descriptor. In some cases this value may be off by one or more words due to timing issues.
7-0 OFFSET	Offset. Local offset field from JUMP Command. This can be any meaningful 8-bit field placed in the JUMP Command. Note that a value of 00x0 will not result in an error being reported.

### B.2.6.9.4 JRSTA Register SSED Field When SSRC = 0100 (DECO Status Source)

When the SSRC value indicates the status source is DECO (SSRC=0100), then bits 27-0 are as shown in [Figure B-109](#).

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JM P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-109. JRSTA Register SSED Field when SSRC = DECO - Format**

**Table B-98. JRSTA Register SSED Field when SSRC = DECO - Field Descriptions**

Field	Description
27 JMP	Jump. The Descriptor made a jump to another Descriptor. When this bit is set, the DESC INDEX field will contain the index into the Descriptor that was jumped to, and not to the original Descriptor.
26	Reserved
25-16	Reserved.

**Table B-98. JRSTA Register SSED Field when SSRC = DECO - Field Descriptions**

Field	Description																																																							
15-8 DESC INDEX	Index into the failing Descriptor. Index is number of words from start of Descriptor. In some cases this value may be off by one or more words due to timing issues.																																																							
7-0 DESC ERROR	Descriptor Error type. <table border="1"> <thead> <tr> <th>Desc Error</th> <th>Details</th> </tr> </thead> <tbody> <tr><td>00h</td><td>None. No error.</td></tr> <tr><td>01h</td><td>SGT Length Error. The descriptor is trying to read more data than is contained in the SGT table.</td></tr> <tr><td>02h</td><td>SGT Null Entry error. Extension bit set but SGT entry was null.</td></tr> <tr><td>03h</td><td>Job Ring Control Error. There is a bad value in the Job Ring Control register.</td></tr> <tr><td>04h</td><td>Invalid Descriptor Command. The Descriptor Command field is invalid.</td></tr> <tr><td>05h</td><td>Reserved.</td></tr> <tr><td>06h</td><td>Invalid KEY Command.</td></tr> <tr><td>07h</td><td>Invalid LOAD Command.</td></tr> <tr><td>08h</td><td>Invalid STORE Command.</td></tr> <tr><td>09h</td><td>Invalid OPERATION Command.</td></tr> <tr><td>0Ah</td><td>Invalid FIFO LOAD Command.</td></tr> <tr><td>0Bh</td><td>Invalid FIFO STORE Command.</td></tr> <tr><td>0Ch</td><td>Invalid MOVE/MOVE_LEN Command.</td></tr> <tr><td>0Dh</td><td>Invalid JUMP Command. A nonlocal JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a Trusted Descriptor to a Job Descriptor, or because the target Descriptor contains a Shared Descriptor.</td></tr> <tr><td>0Eh</td><td>Invalid MATH Command. The MATH Command is invalid.</td></tr> <tr><td>0Fh</td><td>Invalid SIGNATURE Command.</td></tr> <tr><td>10h</td><td>Invalid Sequence Command. A SEQ IN PTR OR SEQ OUT PTR Command is invalid or a SEQ KEY, SEQ LOAD, SEQ FIFO LOAD, or SEQ FIFO STORE decremented the input or Output Sequence length below 0.</td></tr> <tr><td>11h</td><td>Skip data type invalid. The type must be Eh or Fh.</td></tr> <tr><td>12h</td><td>Shared Descriptor Header Error.</td></tr> <tr><td>13h</td><td>Header Error. Invalid length or parity, or certain other problems.</td></tr> <tr><td>14h</td><td>Burster Error. Burster has gotten to an illegal state.</td></tr> <tr><td>16h</td><td>DMA Error.</td></tr> <tr><td>1Ah</td><td>Job failed due to JR reset</td></tr> <tr><td>1Bh</td><td>Job failed due to Fail Mode</td></tr> <tr><td>1Ch</td><td>DECO Watchdog timer timeout error</td></tr> <tr><td>1Fh</td><td>MID error. DECO was trying to share from itself but the two Non-SEQ MID values didn't match or the "shared from" DECO's Descriptor required that the SEQ MIDs be the same and they aren't.</td></tr> </tbody> </table>		Desc Error	Details	00h	None. No error.	01h	SGT Length Error. The descriptor is trying to read more data than is contained in the SGT table.	02h	SGT Null Entry error. Extension bit set but SGT entry was null.	03h	Job Ring Control Error. There is a bad value in the Job Ring Control register.	04h	Invalid Descriptor Command. The Descriptor Command field is invalid.	05h	Reserved.	06h	Invalid KEY Command.	07h	Invalid LOAD Command.	08h	Invalid STORE Command.	09h	Invalid OPERATION Command.	0Ah	Invalid FIFO LOAD Command.	0Bh	Invalid FIFO STORE Command.	0Ch	Invalid MOVE/MOVE_LEN Command.	0Dh	Invalid JUMP Command. A nonlocal JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a Trusted Descriptor to a Job Descriptor, or because the target Descriptor contains a Shared Descriptor.	0Eh	Invalid MATH Command. The MATH Command is invalid.	0Fh	Invalid SIGNATURE Command.	10h	Invalid Sequence Command. A SEQ IN PTR OR SEQ OUT PTR Command is invalid or a SEQ KEY, SEQ LOAD, SEQ FIFO LOAD, or SEQ FIFO STORE decremented the input or Output Sequence length below 0.	11h	Skip data type invalid. The type must be Eh or Fh.	12h	Shared Descriptor Header Error.	13h	Header Error. Invalid length or parity, or certain other problems.	14h	Burster Error. Burster has gotten to an illegal state.	16h	DMA Error.	1Ah	Job failed due to JR reset	1Bh	Job failed due to Fail Mode	1Ch	DECO Watchdog timer timeout error	1Fh	MID error. DECO was trying to share from itself but the two Non-SEQ MID values didn't match or the "shared from" DECO's Descriptor required that the SEQ MIDs be the same and they aren't.
Desc Error	Details																																																							
00h	None. No error.																																																							
01h	SGT Length Error. The descriptor is trying to read more data than is contained in the SGT table.																																																							
02h	SGT Null Entry error. Extension bit set but SGT entry was null.																																																							
03h	Job Ring Control Error. There is a bad value in the Job Ring Control register.																																																							
04h	Invalid Descriptor Command. The Descriptor Command field is invalid.																																																							
05h	Reserved.																																																							
06h	Invalid KEY Command.																																																							
07h	Invalid LOAD Command.																																																							
08h	Invalid STORE Command.																																																							
09h	Invalid OPERATION Command.																																																							
0Ah	Invalid FIFO LOAD Command.																																																							
0Bh	Invalid FIFO STORE Command.																																																							
0Ch	Invalid MOVE/MOVE_LEN Command.																																																							
0Dh	Invalid JUMP Command. A nonlocal JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a Trusted Descriptor to a Job Descriptor, or because the target Descriptor contains a Shared Descriptor.																																																							
0Eh	Invalid MATH Command. The MATH Command is invalid.																																																							
0Fh	Invalid SIGNATURE Command.																																																							
10h	Invalid Sequence Command. A SEQ IN PTR OR SEQ OUT PTR Command is invalid or a SEQ KEY, SEQ LOAD, SEQ FIFO LOAD, or SEQ FIFO STORE decremented the input or Output Sequence length below 0.																																																							
11h	Skip data type invalid. The type must be Eh or Fh.																																																							
12h	Shared Descriptor Header Error.																																																							
13h	Header Error. Invalid length or parity, or certain other problems.																																																							
14h	Burster Error. Burster has gotten to an illegal state.																																																							
16h	DMA Error.																																																							
1Ah	Job failed due to JR reset																																																							
1Bh	Job failed due to Fail Mode																																																							
1Ch	DECO Watchdog timer timeout error																																																							
1Fh	MID error. DECO was trying to share from itself but the two Non-SEQ MID values didn't match or the "shared from" DECO's Descriptor required that the SEQ MIDs be the same and they aren't.																																																							

**Table B-98. JRSTA Register SSED Field when SSRC = DECO - Field Descriptions**

Field	Description														
7-0 DESC ERROR, continued	Descriptor Error type.														

Desc Error	Details
20h	DECO has completed a reset initiated via the DRR register
21h	Nonce error. When using EKT (CCM) key encryption option in the FIFO STORE Command, the Nonce counter reached its maximum value and this encryption mode can no longer be used.
23h	Read Input Frame error. A read input frame was attempted, but the protocol executed does not support it or a SEQ IN PTR command has not been executed.
24h	KEK not loaded error. JDKEK, TDKEK or TDSK was needed, but value has not yet been initialized.
27h	Rewind Error. Trying to do a rewind of the input frame but input buffers have already been released
80h	DNR (do not run) error. A Job Descriptor or Shared Descriptor had the DNR bit set.
81h	undefined protocol command
C1h	Blob Command error: Undefined mode
C2h	Blob Command error: Secure Memory Blob mode error
C4h	Blob Command error: Black Blob key or input size error
C5h	Blob Command error: Invalid key destination
C8h	Blob Command error: Trusted/Secure mode error
All other values are reserved.	

### B.2.6.9.5 JRSTA Register SSED Field When SSRC=0110 (Job Ring Status Source)

When the SSRC value indicates the status source is Job Ring (SSRC=0110), then bits 27-0 are as shown in [Figure B-110](#).

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NADDR									DESC ERROR	

**Figure B-110. JRSTA Register SSED Field when SSRC=Job Ring - Format****Table B-99. JRSTA Register SSED Field when SSRC=Job Ring - Field Descriptions**

Field	Description
27-11	Reserved.

**Table B-99. JRSTA Register SSED Field when SSRC=Job Ring - Field Descriptions**

Field	Description										
10-8 NADDR	Number of Descriptor addresses requested when an error of type 1E (error reading Descriptor address) occurred.										
7-0 DESC ERROR	Descriptor Error. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Desc Error</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td>00h</td> <td>None. No error.</td> </tr> <tr> <td>1Eh</td> <td>Error reading the Descriptor address.</td> </tr> <tr> <td>1Fh</td> <td>Error reading the Descriptor.</td> </tr> <tr> <td colspan="2">All other values are reserved.</td></tr> </tbody> </table>	Desc Error	Details	00h	None. No error.	1Eh	Error reading the Descriptor address.	1Fh	Error reading the Descriptor.	All other values are reserved.	
Desc Error	Details										
00h	None. No error.										
1Eh	Error reading the Descriptor address.										
1Fh	Error reading the Descriptor.										
All other values are reserved.											

### B.2.6.9.6 JRSTA Register SSED Field When SSRC=0111 (Jump Halt Condition Codes Source)

When the SSRC value indicates the status source is Jump Halt Condition Codes (SSRC=0111), then bits 27-0 are as shown in [Figure B-111](#).

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-111. JRSTA Register SSED Field when SSRC= Jump Halt Condition Codes - Format**

**Table B-100. JRSTA Register SSED Field when SSRC= Jump Halt Condition Codes - Field Descriptions**

Field	Description
27 JMP	Jump. The Descriptor made a jump to another Descriptor. When this bit is set, the DESC INDEX field will contain the index into the Descriptor that was jumped to, and not to the original Descriptor.
26-16	Reserved.
15-8 DESC INDEX	Index into the failing Descriptor. Index is number of words from start of Descriptor. In some cases this value may be off by one or more words due to timing issues.
7-0 COND	Condition codes. Condition codes field from JUMP Command.

### B.2.6.10 Job Ring Interrupt Status Register (JRINTR)

The Job Ring Interrupt Status Register indicates whether CAAM has asserted an interrupt for a particular Job Ring, whether software has requested that the Job Ring be halted, whether the Job Ring is now halted, and whether there is an error in this Job Ring. If there was an error, the type of error is indicated. The error bit in the JRINT Register doesn't assert when there is a non-zero job completion status. It only asserts for the types of errors reported in the ERR\_TYPE field in this register. Because there are 2 Job Rings, there are 2 copies of this register. The bit assignments of this register appear in the figure below and the description and settings for the register are given in the following table.

	Access: Read/w1c																																							
Offset	0xBASE_104C (JRINTR0) (used by Job Ring 0)								Accessible only when using MID specified in JR0MIDR register																															
	0xBASE_204C (JRINTR1) (used by Job Ring 1)																																							
ERR_ORWI																																								
PO Reset																																								
W																																								
W																																								
PO Reset																																								

Figure B-112. Job Ring Interrupt Status Register - Format

Table B-101. Job Ring Interrupt Status Register - Field Descriptions

Field	Description
31-30	Reserved.
29-16 ERR_ORWI	Output ring write index with error. Set only when ERR_TYPE=0001. This indicates the location in the Output Ring that was being written when the error occurred. It is the offset in bytes from the Output Ring Base Address (see Section B.2.6.5 on page B-560).
15-12	Reserved.

**Table B-101. Job Ring Interrupt Status Register - Field Descriptions**

Field	Description
11-8 ERR_TYPE	Error type. Set only when JRE bit is also set. Indicates the type of error when it cannot be reported in the Job Ring Status Register (see Section <a href="#">B.2.6.9 on page B-564</a> .) 0b0001: Error writing status to Output Ring 0b0011: Bad Input Ring base address (not on a 4-byte boundary). 0b0100: Bad Output Ring base address (not on a 4-byte boundary). 0b0101: Invalid write to Input Ring Base Address Register. Can be written when there are no jobs in the Input Ring or when the Job Ring is halted. <sup>1</sup> 0b0110: Invalid write to Output Ring Base Address Register. Can be written when there are no jobs in the Output Ring and no jobs from this queue are already processing in CAAM (in the holding tanks or DECOs), or when the Job Ring is halted. <sup>1</sup> 0b0111: Job Ring reset released before Job Ring is halted. <sup>1</sup> 0b1000: Removed too many jobs (ORJRR larger than ORSFR). <sup>1</sup> 0b1001: Added too many jobs (IRJAR larger than IRSAR). <sup>1</sup>  <sup>1</sup> These are fatal and will likely result in not being able to get all jobs out into the Output Ring for processing by software. Resetting the Job Ring will almost certainly be necessary.
3-2 HALT	Halt the Job Ring. If reading HALT returns 01: Software has requested that CAAM flush the jobs in this Job Ring and halt processing jobs in this Job Ring (by writing to the RESET bit in the Job Ring Command register.). If reading HALT returns 10: CAAM has flushed all jobs from this Job Ring and has halted processing jobs in this Job Ring. If there is not enough room in the Output Ring for all the flushed jobs, HALT will continue to return 01 until software has removed enough jobs so that all the flushed jobs can be written to the Output Ring.  Software writes a “1” to the MSB of HALT (bit 3) to clear the HALT field and resume processing jobs in this Job Ring. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that CAAM has flushed all jobs from this Job Ring.  If SNVS indicates a FAIL MODE, a Job Ring halt will be initiated and the HALT status will return 01. When the halt process is complete, the HALT status will be 10. The HALT status cannot be cleared until SNVS transitions out of FAIL MODE.
1 JRE	Job Ring Error. The last Descriptor processed by this Job Ring resulted in an error. The error code is indicated in the Job Ring Output Status Register (see Section <a href="#">B.2.6.9 on page B-564</a> ) and in the Output Ring. Write a 1 to this bit to clear the error indication.
0 JRI	Job Ring Interrupt. CAAM has asserted the interrupt request signal for this Job Ring. Write a 1 to this bit to clear the interrupt request.

### B.2.6.11 Job Ring Configuration Register (JRCFGGR)

Software uses the Job Ring Configuration Register to configure the interrupt handling, error handling, and data endianness specific to a Job Ring. Because there are 2 Job Rings, there are 2 copies of this register. Since there are more than 32 bits in the JRxCFG Register, it is accessed as two 32-bit registers.

Note that many of the bits of this register are used to configure how data is rearranged when it is read from or written to memory. This is intended primarily to facilitate data handling in SoCs in which different processors use different data endianness. Because data may have to be rearranged differently depending upon the type of data, this register provides separate

configuration bits for “control data” and for “message data”. These are defined as shown below:

**Table B-102. Control Data vs. Message Data**

Control Data	Message Data
<p><b>Control data read by CAAM DMA:</b></p> <ul style="list-style-type: none"> <li>• Descriptors or other data loaded into the Descriptor Buffer</li> <li>• Job Ring Input Ring entries</li> <li>• Address pointers</li> <li>• Scatter/Gather Tables</li> <li>• Data loaded into the Class 1 or Class 2 Key Size registers</li> <li>• Data loaded into the Class 1 or Class 2 Data Size registers</li> <li>• Data loaded into the Class 1 or Class 2 ICV Size registers</li> <li>• Data loaded into the CHA Control register</li> <li>• Data loaded into the DECO Control register</li> <li>• Data loaded into the IRQ Control register</li> <li>• Data loaded into the Clear Written register</li> <li>• Data loaded into the Math registers</li> <li>• Data loaded into the AAD Size register</li> <li>• Data loaded into the Class 1 IV Size register</li> <li>• Data loaded into the Alternate Data Size Class 1 register</li> <li>• Data loaded into the iNformation FIFO (NFIFO)</li> </ul>	<p><b>Message data read by CAAM DMA:</b></p> <ul style="list-style-type: none"> <li>• Data read into the Input Data FIFO</li> <li>• Data loaded into the Output Data FIFO</li> <li>• Data loaded into the Class 1 or Class 2 Context registers</li> <li>• Data loaded into the Class 1 or Class 2 Key registers</li> <li>• Data loaded into the Input or Output Data FIFO Nibble Shift registers</li> <li>• Data put into the Auxiliary Data FIFO</li> </ul>

**Table B-102. Control Data vs. Message Data**

Control Data	Message Data
<p><b>Control data written by CAAM DMA:</b></p> <ul style="list-style-type: none"> <li>• Descriptors or other data stored from the Descriptor Buffer</li> <li>• Job Ring Output Ring entries</li> <li>• Address pointers</li> <li>• Scatter/Gather Tables</li> <li>• Data stored from the Class 1 or Class 2 Mode registers</li> <li>• Data stored from the DECO Job Queue Control register</li> <li>• Data stored from the Class 1 or Class 2 Key Size registers</li> <li>• Data stored from the DECO Descriptor Address Register</li> <li>• Data stored from the Class 1 or Class 2 Data Size registers</li> <li>• Data stored from the DECO Status register</li> <li>• Data stored from the Class 1 or Class 2 ICV Size registers</li> <li>• Data stored from the CHA Control register</li> <li>• Data stored from the IRQ Control register</li> <li>• Data stored from the Clear Written register</li> <li>• Data stored from the Math registers</li> <li>• Data stored from the CCB Status register</li> <li>• Data stored from the AAD Size register</li> <li>• Data stored from the Class 1 IV Size register</li> <li>•</li> </ul>	<p><b>Message data written by CAAM DMA:</b></p> <ul style="list-style-type: none"> <li>• Data output via the Output Data FIFO</li> <li>• Data stored from the Class 1 or Class 2 Context registers</li> </ul>

Offset	Access: Read/Write															
	Accessible only when using MID specified in JR0MIDR register															
0xBASE_1050 (JRCFGRO_MS) (used by Job Ring 0)	Accessible only when using MID specified in JR1MIDR register															
0xBASE_2050 (JRCFGRI_MS) (used by Job Ring 1)																
PO Reset	R W															
	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16												
	R	INCL SEQ OUT	0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
	W															
	R	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0											
	W					CHWSO CBSO	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0
	PO Reset	0 0	0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

**Figure B-113. Job Ring Configuration Register\_MS - Format**

Offset	0xBASE_1054 (JRCFGRO_LS) (used by Job Ring 0) 0xBASE_2054 (JRCFGRI_LS) (used by Job Ring 1)	Access: Read/Write Accessible only when using MID specified in JR0MIDR register Accessible only when using MID specified in JR1MIDR register
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	ICTT
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	ICDCT   0 0 0 0   0 0 ICEN IMSK

**Table B-103. Job Ring Configuration Register - Field Descriptions**

Field	Description						
Most-significant half of register							
31							
30 INCL_SEQ_OUT	Include Sequence Out Length. If this bit is set to 1, entries in the Job Ring's Output Ring will include a 32-bit word indicating the number of bytes written out via SEQ STORE and SEQ FIFO STORE commands in this job. If this bit is set to 0, the SEQ OUT Length is omitted from the entries. The setting of this bit can be changed only during ring configuration, when no jobs are running in CAAM, else an error will be flagged.						
29-18	Reserved. Must be 0.						
17 PEO	Platform Endian Override - The bit is XORed with the PLEND bit in the CSTA Register and the other "swap" bits in the Job Ring Configuration Register to determine the AXI Master's view of memory endianness when executing Job Descriptors from this Job Ring. Note that the swap bits can be used in combination to achieve multiple swaps simultaneously.						
16 DMBS	Descriptor Message Data Byte Swap (this applies only to internal message data transfers to/from DECO Descriptor Buffers). An example is shown below:  <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Data as it is read from the source</td> <td style="padding: 2px; color: red;">0123456789abcdefh</td> </tr> <tr> <td style="padding: 2px;">Data as it is written to the destination when DBMS = 0</td> <td style="padding: 2px; color: red;">0123456789abcdefh</td> </tr> <tr> <td style="padding: 2px;">Data as it is written to the destination when DBMS = 1</td> <td style="padding: 2px; color: red;">23016745ab89efcdh</td> </tr> </table>	Data as it is read from the source	0123456789abcdefh	Data as it is written to the destination when DBMS = 0	0123456789abcdefh	Data as it is written to the destination when DBMS = 1	23016745ab89efcdh
Data as it is read from the source	0123456789abcdefh						
Data as it is written to the destination when DBMS = 0	0123456789abcdefh						
Data as it is written to the destination when DBMS = 1	23016745ab89efcdh						
15-14	Reserved. Always 0.						
13 CHWSO	To assist with mixed Endianness platforms, this bit configures a halfword swap of control data written by CAAM DMA. An example is shown below:  <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Data as interpreted within CAAM</td> <td style="padding: 2px; color: red;">0123456789abcdefh</td> </tr> <tr> <td style="padding: 2px;">Data as written to memory when PEO XOR PLEND XOR CHWSO = 0</td> <td style="padding: 2px; color: red;">0123456789abcdefh</td> </tr> <tr> <td style="padding: 2px;">Data as written to memory when PEO XOR PLEND XOR CHWSO = 1</td> <td style="padding: 2px; color: red;">45670123cdef89abh</td> </tr> </table>	Data as interpreted within CAAM	0123456789abcdefh	Data as written to memory when PEO XOR PLEND XOR CHWSO = 0	0123456789abcdefh	Data as written to memory when PEO XOR PLEND XOR CHWSO = 1	45670123cdef89abh
Data as interpreted within CAAM	0123456789abcdefh						
Data as written to memory when PEO XOR PLEND XOR CHWSO = 0	0123456789abcdefh						
Data as written to memory when PEO XOR PLEND XOR CHWSO = 1	45670123cdef89abh						

**Table B-103. Job Ring Configuration Register - Field Descriptions**

Field	Description	
12 CBSO	To assist with mixed Endianness platforms, this bit configures a byte swap of control data written by CAAM DMA. An example is shown below:	
	Data as interpreted within CAAM	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR CBSO = 0	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR CBSO = 1	23016745ab89efcdh
11-10	Reserved. Always 0.	
9 MHWSO	To assist with mixed Endianness platforms, this bit configures a halfword swap of message data written by CAAM DMA. An example is shown below:	
	Data as interpreted within CAAM	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR MHWSO = 0	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR MHWSO = 1	45670123cdef89abh
8 MBSO	To assist with mixed Endianness platforms, this bit configures a byte swap of message data written by CAAM DMA. An example is shown below:	
	Data as interpreted within CAAM	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR MBSO = 0	0123456789abcdefh
	Data as written to memory when PEO XOR PLEND XOR MBSO = 1	23016745ab89efcdh
7-6	Reserved. Always 0.	
5 CHWSI	To assist with mixed Endianness platforms, this bit configures a halfword swap of control data read by CAAM DMA. An example is shown below:	
	Data as stored in memory	0x0123456789abcdefh
	Data as interpreted by CAAM when PEO XOR PLEND XOR CHWSI = 0	0x0123456789abcdefh
	Data as interpreted by CAAM when PEO XOR PLEND XOR CHWSI = 1	0x45670123cdef89abh
4 CBSI	To assist with mixed Endianness platforms, this bit configures a byte swap of control data read by CAAM DMA. An example is shown below:	
	Data as stored in memory	0x0123456789abcdefh
	Data as interpreted by CAAM when PEO XOR PLEND XOR CBSI = 0	0x0123456789abcdefh
	Data as interpreted by CAAM when PEO XOR PLEND XOR CBSI = 1	0x23016745ab89efcdh
3-2	Reserved. Always 0.	

**Table B-103. Job Ring Configuration Register - Field Descriptions**

Field	Description	
1 MHWISI	To assist with mixed Endianness platforms, this bit configures a halfword swap of message data read by CAAM DMA. An example is shown below:	
	Data as stored in memory	0x0123456789abcdef
	Data as interpreted by CAAM when PEO XOR PLEND XOR MHWISI = 0	0x0123456789abcdef
	Data as interpreted by CAAM when PEO XOR PLEND XOR MHWISI = 1	0x45670123cdef89ab
0 MBSI	To assist with mixed Endianness platforms, this bit configures a byte swap of message data read by CAAM DMA. An example is shown below:	
	Data as stored in memory	0x0123456789abcdef
	Data as interpreted by CAAM when PEO XOR PLEND XOR MBSI = 0	0x01234567ababedef
	Data as interpreted by CAAM when PEO XOR PLEND XOR MBSI = 1	0x23016745ab89efcd
Least-significant half of register		
31-16 ICTT	Interrupt Coalescing Timer Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines the maximum amount of time after processing a Descriptor before raising an interrupt. If Descriptors have been processed but the Descriptor count threshold has not been met, an interrupt is raised when the interrupt coalescing timer expires. The interrupt coalescing timer is stopped when the Output Ring Slots Full Register is 0. The timer is reset and stopped once an interrupt has been asserted or whenever the Output Ring Jobs Removed Register is written by software. Counting resumes from zero after a reset if the counter is still enabled. The timer begins counting once the next Descriptor is moved to the Output Ring. Note that it is possible for one or more Descriptors to be moved to the Output Ring after software has read the Output Ring Slots Full Register and before software has written the Output Ring Jobs Removed Register. This would cause the timer to be reset to 0, but still counting. In this situation an interrupt would be raised when the timer expires at the full threshold value (unless the interrupt was raised earlier due to the Descriptor Count Threshold). The threshold value is represented in units equal to 64 CAAM interface clocks. Valid values for this field are from 1 to 65535. A value of 0 results in behavior identical to that when interrupt coalescing is disabled.	
15-8 ICDCT	Interrupt Coalescing Descriptor Count Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines how many Descriptors are completed before raising an interrupt. Valid values for this field are from 0 to 255. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the threshold value is reached each time that a Job Descriptor is completed. A value of 0 is treated in the same manner as a value of 1. The value of ICDCT is ignored if ICEN=0.	
7-3	Reserved.	
2	Reserved	

**Table B-103. Job Ring Configuration Register - Field Descriptions**

Field	Description
1 ICEN	<p>Interrupt Coalescing Enable.</p> <p>0 Interrupt coalescing is disabled. If the IMSK bit is cleared, an interrupt is asserted whenever a job is written to the Output Ring. ICDCT is ignored. Note that if software removes one or more jobs and clears the interrupt but the output rings slots full is still greater than 0 (ORSF &gt; 0), then the interrupt will clear but reassert on the next clock cycle.</p> <p>1 Interrupt coalescing is enabled. If the IMSK bit is cleared, an interrupt is asserted whenever the threshold number of frames is reached (ICDCT) or when the threshold timer expires (ICTT). Note that if software removes one or more jobs and clears the interrupt but the interrupt coalescing threshold is still met (ORSF <math>\geq</math> ICDCT), then the interrupt will clear but reassert on the next clock cycle.</p>
0 IMSK	<p>Interrupt Mask. Mask the interrupt that is associated with the particular processor.</p> <p>0 Interrupt enabled.</p> <p>1 Interrupt masked.</p>

### B.2.6.12 Input Ring Read Index Register (IRRIR)

The Input Ring Read Index Register points to the head of the queue within the Input Ring. At this address there will be a pointer to the next Job Descriptor that CAAM will fetch from this Job Ring. After CAAM reads a Job Descriptor from the Job Ring CAAM increments this register by 4.

The index will be added to the Input Ring Base Address to get the physical address. Because there are 2 Job Rings, there are 2 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRRIR may only be written with a transaction with ns=0.

**Figure B-115. Input Ring Read Index Register - Format**

**Table B-104. Input Ring Read Index Register - Field Descriptions**

Field	Description
31-13	Reserved
12-0 IRRI	Input Ring Read Index.

### B.2.6.13 Output Ring Write Index Register (ORWIR)

The Output Ring Write Index Register points to the tail of the queue within the Output Ring. Because there are 2 Job Rings, there are 2 copies of this register. The Output Ring Write Index Register is added to the Output Ring Base Address Register to get the physical address. At this address CAAM writes a pointer to the last Descriptor that CAAM has processed. At the next entry in the ring CAAM writes the completion status of that Descriptor. Every time that a Descriptor has been processed CAAM increments the value in the Output Ring Write Index Register by the size of the pointer (4 bytes) plus the size of the 4-byte completion status word plus an additional 4 bytes if the INCL\_SEQ\_OUT bit in the JRCRGR is 1. So the increment will be 8if INCL\_SEQ\_OUT=0 and 12if INCL\_SEQ\_OUT=1.

When the Job Ring is allocated to TrustZone SecureWorld, ORWIR may only be written with a transaction with ns=0.

Offset	Access: Read-Only															
	0xBASE_1064 (ORWIR0) (used by Job Ring 0)				Accessible only when using MID specified in JR0MIDR register				0xBASE_2064 (ORWIR1) (used by Job Ring 1)				Accessible only when using MID specified in JR1MIDR register			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0														
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-116. Output Ring Write Index Register - Format**

**Table B-105. Output Ring Write Index Register - Field Descriptions**

Field	Description
31-14	Reserved
13-0 ORWI	Output Ring Write Index. The pointer to the next entry in the Output Ring.

### B.2.6.14 Job Ring Command Register (JRCR)

Software can use this register to issue a flush or reset command to a Job Ring. A flush command is issued by writing a 1 to JRCR.RESET when JRCR.RESET=0. A flush is defined as stalling any jobs currently in the Input Ring and terminating (with an error code) any jobs currently in progress in the holding tank or DECO. The terminated jobs will be written to the Output Ring with a status indicating that they were terminated by a flush request. Note that these flushed jobs will count towards the Interrupt Coalescing Descriptor Count. If there is not sufficient space in the Output Ring for all the flushed jobs, Job Ring flushing will be paused until software has made enough space in the Output Ring. After a flush completes, the halt can be cleared and job processing will resume, or a reset can be requested.

A reset command is issued by writing a 1 to JRCR.RESET when JRCR.RESET=1. A reset command will clear all registers in the Job Ring except the following:

- Input Ring Base Address
- Input Ring Size
- Output Ring Base Address
- Output Ring Size
- Job Ring Configuration.

A reset can be initiated only after a flush has been requested and completed as indicated by the HALT field in the Job Ring Interrupt Status Register. After a reset, job processing will resume when the Input Ring Jobs Added Register is written to indicate that new jobs are available.

Because there are two Job Rings, there are two copies of this register.

Offset	0xBASE_106C (JRCCR0)(used by Job Ring 0)	Access: Write-Only
	0xBASE_206C (JRCCR1)(used by Job Ring 1)	Accessible only when using MID specified in JR0MIDR register Accessible only when using MID specified in JR1MIDR register
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	RESET
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	0 0 0 0

**Figure B-117. Job Ring Command Register - Format****Table B-106. Job Ring Command Register - Field Descriptions**

Field	Description
31-1	Reserved. Always 0.
0 RESET	Reset When RESET is 0, software writes a 1 to RESET to request a flush of the Job Ring. If software wants to initiate a reset of the Job Ring, software writes a 1 to the RESET bit after a flush (when RESET is already 1). The reset will clear the RESET bit and other registers in the Job Ring. If no reset is required, software writes a 1 to the MSB of the HALT field in the Job Ring Interrupt Status Register to cause the Job Ring to resume processing jobs. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that the CAAM has flushed all jobs from this Job Ring.

## B.2.7 Secure Memory Registers

This section describes registers found in CAAM's Secure Memory. The Secure Memory contains Partitions numbered 0 to 7, any of which can contain zero or more fixed-size pages.

### B.2.7.1 Secure Memory Command Register (SMCJRx)

At power up all Secure Memory pages are allocated to Partition 0, and ownership of Partition 0 is assigned to TrustZone nonSecureWorld Job Ring 0 (which is owned by default by the processor that asserts MID 0). Any software entity that owns a Job Ring can claim an available partition by writing to the partition's SMAP register (see Section [B.2.7.3 on page B-587](#)). Once a partition has been claimed by a Job Ring owner, commands related to that partition can be issued by writing to the Secure Memory Command Register within that Job Ring register page. Only the owner of Job Ring x can write to SMCJRx. This register

allows software to issue commands to allocate or de-allocate a page, de-allocate (release ownership of) a partition, or determine which partition is associated with a particular page. But commands issued via a particular Job Ring's SMCJR register will be accepted only for partitions owned by the current Job Ring owner.

Allocating a page associates that page with a partition. Access to that page is now controlled by the associated partition's SMJR and SM2/1JR registers. De-allocation of a page removes the association between the page and the partition, making it available for allocation to another partition. If the partition to which the page was allocated is designated as CSP, the page will be zeroized (i.e. erased) upon de-allocation. De-allocation of a partition removes ownership of the partition and de-allocates all pages allocated to that partition. If the de-allocated partition was designated as CSP, all pages associated with the partition are zeroized. Once de-allocated, other Job Ring owners can claim the partition.

The Page Inquiry Command allows software to specify a page and then read from the Secure Memory Command Status Register the number of the partition associated with that page.

There is one copy of this register per Job Ring, accessed at the addresses shown below.

Access: Write-Only

Offset	0xBASE_10F4 (SMCJR0)(used by Job Ring 0)								Accessible only when using the owner MID in JR0MIDR							
	0xBASE_20F4 (SMCJR1)(used by Job Ring 1)								Accessible only when using the owner MID in JR1MIDR							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PAGE															
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PRTN																
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CMD															
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-118. Secure Memory Command Register - Format

Table B-107. Secure Memory Command Register - Field Descriptions

Field	Description
31-16 PAGE	This is the number of the page to be referenced in field CMD. If the page is not available the command is ignored and an error will be shown in the Secure Memory error status register.
15-12	Reserved.
11-8 PRTN	Partition: When an Allocate Page or De-allocate Partition command is issued, the action is performed on the partition indicated in the PRTN field. The PRTN field is ignored for all other commands. If the command is issued via a Job Ring register page but that Job Ring does not own the partition, the command is ignored and an error will be shown in the Secure Memory Command Status Register. But note that if a partition is owned by TrustZone SecureWorld, it can be de-allocated or pages allocated to it only by TrustZone SecureWorld, i.e. when ns=0 on the write to SMCR.

Field	Description
7-4	Reserved.
3-0 CMD	<p>Command:</p> <p>1h: Allocate Page - This command allocates the page specified in the PAGE field to the partition specified in the PRTN field. If the page does not exist or is already allocated to a partition the page will not be allocated and instead a page availability error will be returned. If the partition does not exist or is not owned<sup>1</sup> by the requestor the page will not be allocated and instead a partition ownership error will be returned.</p> <p>2h: De-allocate Page - This command de-allocates the page specified in the PAGE field and returns the page to the pool of available pages. The PRTN field is ignored. If the page is allocated to a partition that is marked as CSP, the page is zeroized before it is returned to the pool. If the page is allocated to a partition that is marked PSP (whether or not it is also marked as CSP) the page will not be de-allocated or zeroized and instead a PSP error will be returned. If the page is not allocated to a partition, is non-existent, or is not yet initialized or zeroized, the page will not be de-allocated and instead a page availability error will be returned. If the page is allocated to a partition that is not owned<sup>1</sup> by the requestor the page will not be de-allocated and instead a partition ownership error will be returned.</p> <p>3h: De-allocate Partition - This command de-allocates the partition specified in the PRTN field and releases all of the partition's pages to the pool of available pages. The PAGE field is ignored. If the partition is marked as CSP, the pages are zeroized before they are returned to the pool. If the partition is marked as PSP (whether or not it is also marked as CSP) the partition will not be de-allocated and its pages will not be released or zeroized and instead a PSP error will be returned. If the partition is not owned<sup>1</sup> by the requestor the partition will not be de-allocated and its pages will not be released or zeroized and instead a partition ownership error will be returned.</p> <p>5h: Page Inquiry - This command indicates the partition to which the page specified in the PAGE field is currently allocated. The PRTN field is ignored. The SMCS register's PO field will show the page allocation status. If the page is allocated to a partition, the SMCS register's PRTN field indicate the partition to which the page is allocated. .</p> <p>All other values reserved</p> <p><sup>1</sup> Each allocated partition is owned by some Job Ring, and the Job Ring's JRMID.JROWN field indicates which MID is associated with that owner. So a partition is considered to be owned by the requestor if the MID used during the write to the SMC register matches the JRMID.JROWN field of the SMC register's Job Ring page.</p>

### B.2.7.2 Secure Memory Command Status Register (SMCSJRx)

This register provides information on the state of the allocation engine and feedback for Allocation Commands. There is one copy of this register for each Job Ring owner. Only the owner of Job Ring x can read from SMCSJRx. Note that some commands (e.g. deallocation of a CSP partition or page) may take many clock cycles to complete. If the same Job Ring owner issues a second command before its previous command has completed, the CERR field will return a “Command Overflow” indication and the second command will be ignored. However, if a command is issued while Secure Memory is completing another Job Ring owner’s command, the second command will be queued and will execute in its turn.

## CAAM Register Descriptions

There is one copy of this register per Job Ring, accessed at the addresses shown below.

Access: Read-Only

Offset	0xBASE_10FC (SMCSJR0)(used by Job Ring 0)	Accessible only when using the owner MID in JR0MIDR
	0xBASE_20FC (SMCSJR1)(used by Job Ring 1)	Accessible only when using the owner MID in JR1MIDR
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	PAGE
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	PRTN
W	CERR AERR   0 0 0 0   PO 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-119. Secure Memory Command Status Register - Format**

**Table B-108. Secure Memory Command Status Register - Field Descriptions**

Field	Description	
31-28	Reserved. Unused bits; Always zero.	
27-16 PAGE	Page. Following a Page Inquiry, Page Allocate or Page De-Allocate Command, this is the Page number specified in the most recently accepted command. That command may not have completed yet. Commands that cause a Command Overflow are not accepted. Following a Partition De-Allocate Command, the value in this field is invalid.	
15-14 CERR	Command Error. If the command issued via the SMCJR register yielded an error, this field shows the error code.	
Value	Description	
00	No Error.	
01	Command has not yet completed.	
10	A security failure occurred.	
11	Command Overflow. Another command was issued by the same Job Ring owner before the owner's previous command completed. The additional command was ignored.	

**Table B-108. Secure Memory Command Status Register - Field Descriptions**

Field	Description																			
13-12 AERR	Allocation Error. If the command issued via the SMCJR register resulted in an allocation error, this field shows the error code. The value in this field will be 00 following a Page Inquiry Command.																			
	<table border="1" data-bbox="295 350 1437 1157"> <thead> <tr> <th data-bbox="295 350 425 401">Value</th><th colspan="2" data-bbox="425 350 915 401">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="295 401 425 494"></td><td colspan="2" data-bbox="425 401 915 494">After an Allocate Page Command</td></tr> <tr> <td data-bbox="295 494 425 566">00</td><td colspan="2" data-bbox="425 494 915 566">After a De-Allocate Page or a De-Allocate Partition Command</td></tr> <tr> <td data-bbox="295 566 425 777">01</td><td data-bbox="425 566 915 777">Reserved.</td><td data-bbox="915 566 1437 777">PSP error. The entity that issued the command owns the partition, but the partition or page cannot be de-allocated because the partition is marked as a PSP partition. If the same request results in a PSP error and a page availability error (because the page is not yet initialized or zeroized), the PSP error code will be returned.</td></tr> <tr> <td data-bbox="295 777 425 895">10</td><td data-bbox="425 777 915 895">Page availability error. The page is either already allocated to a partition, is non-existent, or is not yet initialized.</td><td data-bbox="915 777 1437 895">Page availability error. The page is either not allocated to a partition, is non-existent, or is not yet initialized or zeroized.</td></tr> <tr> <td data-bbox="295 895 425 1157">11</td><td data-bbox="425 895 915 1157">Partition ownership error. The partition specified in the Allocate Page Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and a page availability error, the partition ownership error code will be returned.</td><td data-bbox="915 895 1437 1157">Partition ownership error. The partition specified in the De-Allocate Partition Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and either a page availability error or a PSP error, the partition ownership error code will be returned.</td></tr> </tbody> </table>		Value	Description			After an Allocate Page Command		00	After a De-Allocate Page or a De-Allocate Partition Command		01	Reserved.	PSP error. The entity that issued the command owns the partition, but the partition or page cannot be de-allocated because the partition is marked as a PSP partition. If the same request results in a PSP error and a page availability error (because the page is not yet initialized or zeroized), the PSP error code will be returned.	10	Page availability error. The page is either already allocated to a partition, is non-existent, or is not yet initialized.	Page availability error. The page is either not allocated to a partition, is non-existent, or is not yet initialized or zeroized.	11	Partition ownership error. The partition specified in the Allocate Page Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and a page availability error, the partition ownership error code will be returned.	Partition ownership error. The partition specified in the De-Allocate Partition Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and either a page availability error or a PSP error, the partition ownership error code will be returned.
Value	Description																			
	After an Allocate Page Command																			
00	After a De-Allocate Page or a De-Allocate Partition Command																			
01	Reserved.	PSP error. The entity that issued the command owns the partition, but the partition or page cannot be de-allocated because the partition is marked as a PSP partition. If the same request results in a PSP error and a page availability error (because the page is not yet initialized or zeroized), the PSP error code will be returned.																		
10	Page availability error. The page is either already allocated to a partition, is non-existent, or is not yet initialized.	Page availability error. The page is either not allocated to a partition, is non-existent, or is not yet initialized or zeroized.																		
11	Partition ownership error. The partition specified in the Allocate Page Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and a page availability error, the partition ownership error code will be returned.	Partition ownership error. The partition specified in the De-Allocate Partition Command does not exist or the entity issuing the command does not own the partition. If the same request results in a partition ownership error and either a page availability error or a PSP error, the partition ownership error code will be returned.																		
11-8	Reserved. Must be 0.																			
7-6 PO	Page Owner: Following a Page Inquiry Command, this field indicates if the Page is owned by the entity that issued the inquiry, owned by another entity, or unowned. The value in this field is invalid following any other command issued via the SMCJRx register.																			
	<table border="1" data-bbox="295 1332 1437 1727"> <thead> <tr> <th data-bbox="295 1332 425 1383">Value</th><th colspan="2" data-bbox="425 1332 1437 1383">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="295 1383 425 1501">00</td><td colspan="2" data-bbox="425 1383 1437 1501">Available; Un-owned: The entity that issued the inquiry may allocate this page to a partition. No zeroization is needed since it has already been cleared, therefore no interrupt should be expected.</td></tr> <tr> <td data-bbox="295 1501 425 1573">01</td><td colspan="2" data-bbox="425 1501 1437 1573">Page does not exist in this version or is not initialized yet.</td></tr> <tr> <td data-bbox="295 1573 425 1645">10</td><td colspan="2" data-bbox="425 1573 1437 1645">Another entity owns the page. This page is unavailable to the issuer of the inquiry.</td></tr> <tr> <td data-bbox="295 1645 425 1727">11</td><td colspan="2" data-bbox="425 1645 1437 1727">Owned by the entity making the inquiry. The owner may de-allocate this page if its partition is not marked PSP. If the partition to which the page is allocated is designated as CSP, the page will be zeroized upon de-allocation.</td></tr> </tbody> </table>		Value	Description		00	Available; Un-owned: The entity that issued the inquiry may allocate this page to a partition. No zeroization is needed since it has already been cleared, therefore no interrupt should be expected.		01	Page does not exist in this version or is not initialized yet.		10	Another entity owns the page. This page is unavailable to the issuer of the inquiry.		11	Owned by the entity making the inquiry. The owner may de-allocate this page if its partition is not marked PSP. If the partition to which the page is allocated is designated as CSP, the page will be zeroized upon de-allocation.				
Value	Description																			
00	Available; Un-owned: The entity that issued the inquiry may allocate this page to a partition. No zeroization is needed since it has already been cleared, therefore no interrupt should be expected.																			
01	Page does not exist in this version or is not initialized yet.																			
10	Another entity owns the page. This page is unavailable to the issuer of the inquiry.																			
11	Owned by the entity making the inquiry. The owner may de-allocate this page if its partition is not marked PSP. If the partition to which the page is allocated is designated as CSP, the page will be zeroized upon de-allocation.																			

**Table B-108. Secure Memory Command Status Register - Field Descriptions**

Field	Description
5-4	Reserved. Must be 0.
3-0 PRTN	Following a Page Inquiry Command, if the PO field is 10 or 11, this field indicates the partition to which the page specified in the PAGE field is allocated. Following any other SMCJRx register command, or if the PO field is 00 or 01 following a Page Inquiry Command, the value in this field is invalid.

### B.2.7.3 Secure Memory Access Permissions Register (SMJR)

There is one Secure Memory Access Permissions (SMAPJR) register for each Secure Memory partition, and each register is aliased to multiple addresses, one for each Job Ring. If a partition is not already claimed, a Job Ring owner can claim that partition by writing to the partition's SMAPJR register using the address alias that is located in the Job Ring's register page. That is, the owner of Job Ring 0 would write to the address alias in register page 1 to claim a partition, but the owner of Job Ring 1 would write to the address alias in register page 2 to claim that same partition. Writing to the SMAPJR register also sets the access permissions for the partition. After a partition has been claimed, only the partition owner can change the access permissions for the partition. This is enforced by comparing the MID value asserted during the write to the SMAPJR or SMAGJR register to the JROWN field within the appropriate JRxMID register. If the partition is released, the partition's SMAPJR and SMAG2/1JR registers once again become accessible to any Job Ring Owner.

The access permissions set in the SMAPJR register apply to accesses to the partition made by the bus masters specified via the partition's Secure Memory(SMAG2/1JR) registers. All bus masters not specified via the SMAG2/1JR registers are prohibited from accessing that partition.

Access permissions for a Secure Memory partition are specified by listing the access types that a specified group of bus masters is allowed to make when accessing that partition. There are two separate access groups specified via the SMAG2JR and SMAG1JR registers, and corresponding to these are two separate access permission fields in the SMAPJR register. A particular bus master may belong to either Group 1, or Group 2, or both, or neither. If a bus master belongs to neither group, then that bus master is prohibited from accessing that partition. If the bus master belongs to one of the two access groups, that bus master is allowed to access the partition using the access types permitted for that access group. If the bus master belongs to both groups, the bus master will be able to use the access types specified for either access group.

The SMAPJR register's "Gx\_WRITE" and "Gx\_READ" permission bits are straightforward. Setting these bits allows the members of Group x to write or read

(respectively) that partition either via the AXI Slave Interface, or via descriptor commands that access Secure Memory via CAAM’s internal DMA.

CAAM’s Secure Memory also enforces access permissions with respect to CAAM-only access types. The CAAM-only access types include Trusted Descriptor accesses, key-reads, and Secure Memory Blob accesses. These access types are communicated via internal CAAM signals, so no external bus transactions can indicate these access types.

When Trusted Descriptors access Secure Memory, these accesses receive special treatment. If an access from a Trusted Descriptor uses a MID that is a member of the partition’s access group 1 or access group 2, the access is permitted regardless of the SMAPJR register settings. Since such accesses are always allowed, there is no access control bit associated with Trusted Descriptor access to Secure Memory. Because Trusted Descriptors are allowed unrestricted access to any Secure Memory partition, Trusted Descriptors can be used to create or modify keys in key partitions, or to reload cryptographic keys from blobs.

Note that most accesses to Secure Memory made by Job Descriptors and Shared Descriptors are governed by the G1 and G2 READ and WRITE permissions, but “key-read accesses” initiated by descriptors are not controlled by these permission bits. Key-read accesses are read transactions initiated by a KEY command with NWB=1, no SGF, and length greater than or equal to 16 bytes. These accesses receive special treatment because the data read receives special protection. The only possible destination for the data read via a KEY command is a key register (Class 1 Key Register, Class 2 Key Register, or AFHA S-Box). Once data has been loaded into a key register, CAAM protects the data’s confidentiality by either preventing the data from being transferred back out of the key register (if NWB=1), or by encrypting the data before it is transferred out (if NWB=0).

CAAM grants special permissions for key-read transactions. A key-read access is permitted if an ordinary read is permitted (i.e. if  $Gx\_READ=1$  for one or both of the groups to which the read transaction’s MID belongs), but key-read accesses are also permitted even if  $Gx\_READ=0$ , as long as the MID asserted by the descriptor is a member of Groupx. This mechanism allows a partition to be dedicated for cryptographic keys that are unreadable by software, but which can still be used as keys in CAAM’s CHAs. A “key partition” can be created by setting a partition’s access permissions to prevent ordinary read or write accesses or blob operations. A key partition must be provisioned in a manner that prevents software from knowing the values of the keys stored in the key partition. As mentioned earlier, this can be accomplished via Trusted Descriptors. As an alternative, the SMAPJR and SMAG2/1JR register permissions could be set to allow trusted software to read and write the partition, but prevent untrusted software from accessing the partition except via

key-reads. [Figure B-120. on page B-589](#) indicates the various read and key-read permissions that can be defined.

	G1_READ=1	G1_READ=0
G2_READ=1	read or key-read permitted if MID is a member of G2 read or key-read permitted if MID is a member of G1	read or key-read permitted if MID is a member of G2 key-read permitted if MID is a member of G1
G2_READ=0	key-read permitted if MID is a member of G2 read or key-read permitted if MID is a member of G1	key-read permitted if MID is a member of G2 key-read permitted if MID is a member of G1

\* Note that a read operation is considered a “key-read” only if issued by a KEY command with NWB=1, and only if the access does not use a SGT, and only if the length of the key is 16 bytes or more. Key-read accesses are disallowed if the DECO is under direct control of software.

**Figure B-120. Read and Key-Read Permissions**

If data within a partition must be preserved across power cycles, the data can be cryptographically protected by encapsulating it as a blob before storing it in nonvolatile memory. On a subsequent power cycle (or even during the current power cycle) the data can be recovered by decapsulating the blob. Exporting and importing General Memory Blobs are considered as “reads” and “writes” respectively, and exporting them from or importing them to Secure Memory is permitted or prohibited by the appropriate settings of the “Gx\_READ” and “Gx\_WRITE” permission bits. But if the partition does not allow read access, or if the partition’s access permissions or Job Ring ownership must be enforced, or the blob must be re-imported into a partition that does not allow write access, then a Secure Memory Blob should be used instead of a General Memory Blob.

A Secure Memory Blob can be exported from or imported to a Secure Memory partition by a Job Descriptor (or a Shared Descriptor) only if the partition’s “Gx\_SMBLOB” bit is a 1. Secure Memory Blob operations are not restricted by the Gx\_READ and Gx\_WRITE bits, whereas General Memory Blob operations are not restricted by the “Gx\_SMBLOB” bit. Note that Trusted Descriptors are exempt from all these access restrictions, and can export or import General Memory Blobs or Secure Memory Blobs regardless of the “Gx\_READ”, “Gx\_WRITE” or “Gx\_SMBLOB” settings.

The Blob Key Encryption Key for Secure Memory Blobs is derived, in part, from the values in the partition’s SMAPJR and SMAG2/1JR registers, therefore a Secure Memory Blob can be successfully imported into a partition only if the partition’s SMAPJR and SMAG2/1JR registers have the same values as the partition from which the Secure Memory Blob was exported.<sup>1</sup> If the values differ at all the BKEK value will be incorrect, which will cause the import operation to halt with an integrity check error.

1. Observe that the partition owner’s Job Ring number appears in the SMAPJR register. This supports data privacy by preventing a Secure Memory Blob from being imported into a partition owned by a different Job Ring owner.

The semantics of the access control bits can be summarized in the following Boolean equations:

```

Access_Permitted =
  ( MID ∈ { ACCESS_GROUP1 } ) &
    ( key_read | trusted_descriptor |
      ( sm_blob_access & G1_SMBLOB ) |
      ( write & G1_WRITE ) |
      ( read & G1_READ ) )
  ( MID ∈ { ACCESS_GROUP2 } ) &
    ( key_read | trusted_descriptor |
      ( sm_blob_access & G2_SMBLOB ) |
      ( write & G2_WRITE ) |
      ( read & G2_READ ) )

```

CAPITAL LETTERS represent SMAPJR or SMAG2/1JR registers fields  
lower case letters represent hardware signals

When software sets the access permissions for a partition, the software can mark the partition with a Critical Security Parameters<sup>1</sup> (CSP) flag, which tells Secure Memory that the partition will be used to hold disclosure-sensitive data. To protect this disclosure-sensitive data, the Secure Memory automatically zeroizes CSP partitions when a security alarm is detected, or when releasing that memory for other purposes. Whenever a page is de-allocated from a CSP partition, that page will be zeroized. When a CSP partition is de-allocated the Secure Memory hardware zeroizes all pages in that partition. Not flagging the partition as CSP indicates that the partition will not be used to store disclosure-sensitive data, so the Secure Memory does not zeroize the pages in that partition when CAAM transitions to Fail Mode or when the partition or pages from the partition are de-allocated.

There may be circumstances in which data must remain accessible despite the occurrence of security failures, software errors, or malicious software. When software sets the access permissions for a partition, the software can mark the partition with a Public Security Parameters (PSP) flag, which tells Secure Memory that the partition will be used to hold availability-sensitive data. Flagging a partition as PSP prevents the partition or any of its pages from being de-allocated. If a partition is flagged as both PSP and CSP, the partition (or individual pages of the partition) cannot be de-allocated, but the pages of the partition will be zeroized when CAAM transitions to Fail Mode.

1. The terms “Critical Security Parameter” and “Public Security Parameter” are defined in FIPS 140-3.

Normally the owner of a partition can change the values in the partition's SMAPJR and SMAG2/1JR registers at any time, provided that the owner still has control of a Job Ring. But the changes to either the SMAPJR register or the SMAG2JR and SMAG1JR registers can be prevented by setting the "SMAPJR\_LCK" or "SMAG\_LCK" bits, respectively. Note that once one of these LCK bits has been set, it cannot be unlocked except by a POR or by de-allocating the partition. This mechanism allows either the access permissions or the group membership, or both, to be set by trusted software before ownership of the partition is turned over to less trusted software.

Note that at power on all Secure Memory pages are automatically allocated to partition 0, which is owned by Job Ring 0, and the access control bits of partition 0's SMAPJR register and the group membership bits of partition 0's SMAG2/1JR registers reset to values that make partition 0 accessible to any MID via any access mode. Similarly the lock bits reset to 0 on power on so that there are no restrictions on changing the SMAPJR and SMAG2/1JR values. Also, the CSP and PSP bits reset to 0 so that the associated partition is not marked as either a critical security parameter or a public security parameter.

There is only one copy of this register per partition, but that copy is accessible at multiple addresses, one for each Job Ring. In SMAPJR\_<p>, JR represents the Job Ring number and <p> represents the partition number. "offset 16" means that the registers are located 16 bytes apart (starting at x104). "range 0..7" defines the upper and lower limits for <p>.

																Access: Read/Write If the partition is allocated, SMAPJR is accessible only by the partition owner															
Offset		PARTITION_KMOD																													
R																															
W																															
PO Reset		0xBASE_1104 (SMAPJR0_<p>), offset 16, range p=0..7				0xBASE_2104 (SMAPJR1_<p>), offset 16, range p=0..7				Accessible only when using the owner MID in JR0MIDR				Accessible only when using the owner MID in JR1MIDR																	
R																															
W		CSP				PSP		SMAP _LCK		SMAG _LCK		JR		0		0		G2_ SMBLOB													
PO Reset		0				0		0		0		0		0		0		RSV													
R																		G2_ WRITE													
W								w1s		w1s								G2_ READ													
PO Reset		0				0		0		0		0		1		1		1													
R								7		6		5		4		3		2													
W								1		1		1		1		1		1													
PO Reset		0				0		0		0		0		0		1		1													
R								G1_ SMBLOB		RSV		G1_ WRITE		G1_ READ																	
W								1		1		1		1		1															
PO Reset		0				0		0		0		0		0		1		1													

Figure B-121. Access Permissions Register - Format

**Table B-109. Access Permissions Register - Field Descriptions**

Field	Description
31-16 PARTITION_KMOD	The value in this field is used to modify the Blob Key Encryption Key when exporting cryptographic Blobs from, or importing cryptographic Blobs to, this partition. The value can be chosen by software to be unique for each partition, ensuring that Blobs will not decrypt correctly if imported into the wrong partition. The PARTITION_KMOD value can also be used a version ID, ensuring the Blob will not decrypt correctly if the wrong version is imported.
15 CSP	<p>Critical Security Parameters. This field determines whether the pages from the partition will be zeroized when they are de-allocated or a security alarm occurs.</p> <p>0 The pages allocated to the partition will <b>not</b> be zeroized when they are de-allocated or the partition is released or a security alarm occurs.</p> <p>1 The pages allocated to the partition <b>will</b> be zeroized when they are individually de-allocated or the partition is released or a security alarm occurs.</p>
14 PSP	<p>Public Security Parameters. This field determines whether the partition and pages from the partition can be de-allocated.</p> <p>0 - The partition and any of the pages allocated to the partition can be de-allocated.</p> <p>1 - The partition <b>cannot</b> be de-allocated and the pages allocated to the partition <b>cannot</b> be de-allocated.</p>
13 SM_LCK	<p>SM LOCK bit. This field determines whether changes can be made to the partition's access control permissions.</p> <p>0 - The SMJR register is unlocked. The partition owner can change any writable bits of the SMJR register.</p> <p>1 - The SMJR register is locked. The SM_LCK, CSP and PSP bits and G1 and G2 permission bits of the SMJR register cannot be changed until the partition is de-allocated or a POR occurs. The PARTITION_KMOD value can still be changed. The SMAG_LCK bit can be changed to a 1, but cannot be changed to a 0.</p>
12 SMAG_LCK	<p>SMAG LOCK bit. This field determines whether changes can be made to the partition's access group definitions.</p> <p>0 - The SMAG2JR register and SMAG1JR register are unlocked. The partition owner can change any writable bits of these registers.</p> <p>1 - The SMAG2JR register and SMAG1JR register are locked. The SMAG2JR and SMAG1JR registers cannot be changed until the partition is de-allocated or a POR occurs.</p>
11-10 JR	<p>Job Ring number. Used when deriving a BKEK for Blob export or import. Cryptographically binds a Blob to a particular Job Ring.</p> <p>00 JR0 01 JR1</p>
9-8	Reserved. Must be 0.
7 G2_SMBLOB	<p>Access Group 2 Secure Memory Blobs. The bus masters whose MID bits are 1 in the partition's SMAG2JR register are allowed to export Secure Memory Blobs from or import Secure Memory Blobs into the partition in accordance with the restrictions specified below. (The setting of this bit does not affect General Memory Blobs. These can be exported if G2_READ=1, and can be imported if G2_WRITE=1.)</p> <p>0 Exporting or importing Secure Memory Blobs is prohibited, unless done via a Trusted Descriptor.</p> <p>1 Exporting or importing Secure Memory Blobs is allowed, regardless of the G2_READ and G2_WRITE settings.</p>
6	Reserved. Can be written to a 0 or 1. Value affects the BKEK derivation.
5 G2_WRITE	<p>Access Group 2 Write. The bus masters whose MID bits are 1 in the partition's SMAG2JR register are allowed to write into the partition in accordance with the restrictions specified below.</p> <p>0 Writes are prohibited (except that Trusted Descriptor writes are allowed, and importing Secure Memory Blobs is allowed if G2_SMBLOB=1 or if done by a Trusted Descriptor).</p> <p>1 Writes are allowed (but importing a Secure Memory Blob is prohibited if G2_SMBLOB=0 and the descriptor is not a Trusted Descriptor).</p>

**Table B-109. Access Permissions Register - Field Descriptions**

Field	Description
4 G2_ READ	Access Group 1 Read. The bus masters whose MID bits are 1 in the partition's SMAG1JR registerare allowed to read or fetch instructions from the partition in accordance with the restrictions specified below. 0 Instruction fetches and reads are prohibited (except that Trusted Descriptor reads and key-reads are always allowed, and exporting Secure Memory Blobs is allowed if G1_SMBLOB=1 or if done by a Trusted Descriptor). 1 Instruction fetches and reads are allowed (but exporting a Secure Memory Blob is prohibited if G2_SMBLOB=0 and the descriptor is not a Trusted Descriptor).
3 G1_ SMBLOB	Access Group 1 Secure Memory Blobs. The bus masters whose MID bits are 1 in the partition's SMAG1JR registerare allowed to export Secure Memory Blobs from or import Secure Memory Blobs into the partition in accordance with the restrictions specified below. (The setting of this bit does not affect General Memory Blobs. These can be exported if G1_READ=1, and can be imported if G1_WRITE=1.) 0 Exporting or importing Secure Memory Blobs is prohibited, unless done via a Trusted Descriptor. 1 Exporting or importing Secure Memory Blobs is allowed, regardless of the G1_READ and G1_WRITE settings.
2	Reserved. Can be written to a 0 or 1. Value affects the BKEK derivation.
1 G1_WRITE	Access Group 1 Write. The bus masters whose MID bits are 1 in the partition's SMAG1JR registerare allowed to write into the partition in accordance with the restrictions specified below. 0 Writes are prohibited (except that Trusted Descriptor writes are allowed, and importing Secure Memory Blobs is allowed if G1_SMBLOB=1 or if done by a Trusted Descriptor). 1 Writes are allowed (but importing a Secure Memory Blob is prohibited if G1_SMBLOB=0 and the descriptor is not a Trusted Descriptor).
0 G1_READ	Access Group 1 Read. The bus masters whose MID bits are 1 in the partition's SMAG1JR registerare allowed to read or fetch instructions from the partition in accordance with the restrictions specified below. 0 Instruction fetches and reads are prohibited (except that Trusted Descriptor reads and key-reads are always allowed, and exporting Secure Memory Blobs is allowed if G1_SMBLOB=1 or if done by a Trusted Descriptor). 1 Instruction fetches and reads are allowed (but exporting a Secure Memory Blob is prohibited if G1_SMBLOB=0 and the descriptor is not a Trusted Descriptor).

### B.2.7.4 Secure Memory Access Groups Registers (SMAG2/1JR)

For each allocated partition, the Secure Memory Access Groups Register indicates which bus masters are allowed to access the partition. The SMAG2JR and SMAG1JR registers define two access groups, ACCESS GROUP 1 and ACCESS GROUP 2, corresponding to the two access permission sets defined in the SMAPJR register. The 32-bit ACCESS GROUP fields are defined so that each bit corresponds to a particular bus MID. For example, the least-significant bit of each ACCESS\_GROUP field represents MID 0b00000, and the most-significant bit represents MID 0b11111. Setting a bit to a 1 indicates that the bus master identified by that particular 5-bit value is a member of that ACCESS GROUP. This encoding allows any combination of 32 bus master identifiers to be included in either ACCESS GROUP.

The nature of the 5-bit bus master identifier associated with an access via the AXI Slave interface is determined by the particular hardware signals that are connected to CAAM's AXI and IP bus interfaces.

The interpretation of bus master identities in the i.MX6 series processors is shown in the figure below. Note that only the least-significant 4 bits of the master ID are used in this encoding.

0000 -	TrustZone SecureWorld, Other DMA
0001 -	TrustZone SecureWorld, ARM
0010 -	TrustZone SecureWorld, CAAM
0011 -	TrustZone SecureWorld, SDMA
1000 -	TrustZone NonsecureWorld, Other DMA
1001 -	TrustZone NonsecureWorld, ARM
1010 -	TrustZone NonsecureWorld, CAAM
1011 -	TrustZone NonsecureWorld, SDMA

**Figure B-122. Bus Master Identities**

For those bus masters that are a member of an ACCESS GROUP, the partition's SMAPJR Register will be consulted to determine which types of access are allowed. If the bus master is a member of ACCESS GROUP 1, the "G1" permission bits will be consulted. If the bus master is a member of ACCESS GROUP 2, the "G2" permission bits will be consulted. If the bus master is a member of both ACCESS GROUPS, the bus master will be allowed any access type permitted by either set of permission bits. Note that all SMAG2/1JR registers reset to all 1s at power on so that by default all bus master identities are members of both ACCESS GROUPS for each Secure Memory partition. Similarly, at reset all SMAPJR registers reset to all access types permitted for both access permission sets.

When CAAM executes a descriptor that causes any memory transactions (even internal accesses to Secure Memory), CAAM will assert the MID value specified in the JRDMA field within the appropriate JRxMID register (where x is the number of the Job Ring from which the descriptor was executed). The values in these fields can be set so that CAAM will assert a different MID when performing DMA operations on behalf of jobs drawn from different Job Rings. If so, then jobs from different Job Rings can be permitted or denied access to particular Secure Memory partitions by setting the partition's SMAG2/1JR registers appropriately.

The values in the SMAG2JR\_p and SMAG1JR\_p registers have no effect if partition p is currently unallocated. See Section [B.2.7.3 on page B-587](#) for an explanation of how a partition is allocated. Once partition p has been allocated to Job Ring Owner x, the SMAPJRx\_p and SMAG2/1x\_p registers become accessible only to Job Ring Owner x (i.e., a bus master using the identity that is currently in the JROWN field of the Job Ring's JRxMID register). Attempts to write to the SMAPJR or SMAG2/1JR registers of a partition owned by another MID will be ignored, and read transactions from these registers will

## CAAM Register Descriptions

return all 0s. If the partition is released, the partition's SMAPJR and SMAG2/1JR registers once again become accessible (at the address alias appropriate for that Job Ring) to all Job Ring Owners on a first-come first-served basis.

There is only one copy of this register per partition, but that copy is accessible at multiple addresses, one for each Job Ring. In SMAG<sub>x</sub>\_<i>, x represents the Job Ring number and <i> represents the partition number. "offset 16" means that the registers are located 16 bytes apart (starting at x10C). "range 0..7" defines the upper and lower limits for <p>.

Offset																Access: Read/Write If the partition is allocated, SMAG2JR is accessible only by the partition owner			
0xBASE_1108 (SMAG2JR0_<p>), offset 16, range p=0..7 0xBASE_2108 (SMAG2JR1_<p>), offset 16, range p=0..7																Accessible only when using the owner MID in JR0MIDR Accessible only when using the owner MID in JR1MIDR			
R    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0																R    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
W    0																W    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
PO Reset 0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0																PO Reset 0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
R    G2_M15    G2_M14    G2_M13    G2_M12    G2_M11    G2_M10    G2_M9    G2_M8    G2_M7    G2_M6    G2_M5    G2_M4    G2_M3    G2_M2    G2_M1    G2_M0																R    G2_M15    G2_M14    G2_M13    G2_M12    G2_M11    G2_M10    G2_M9    G2_M8    G2_M7    G2_M6    G2_M5    G2_M4    G2_M3    G2_M2    G2_M1    G2_M0			
W    1																W    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1			
PO Reset 1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1																PO Reset 1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1			

**Figure B-123. Secure Memory Access Groups Register, Group 2 - Format**

There is only one copy of this register per partition, but that copy is accessible at multiple addresses, one for each Job Ring. In SMAG1JR<sub>x</sub>\_<p>, x represents the Job Ring number and <p> represents the partition number. "offset 16" means that the registers are located 16 bytes apart (starting at x10C). "range 0..7" defines the upper and lower limits for <7>.

Offset																Access: Read/Write If the partition is allocated, SMAG1JR is accessible only by the partition owner			
0xBASE_110C (SMAG1JR0_<p>), offset 16, range p=0..7 0xBASE_210C (SMAG1JR1_<p>), offset 16, range p=0..7																Accessible only when using the owner MID in JR0MIDR Accessible only when using the owner MID in JR1MIDR			
R    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0																R    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
W    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0																W    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
PO Reset 0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0																PO Reset 0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0			
R    G1_M15    G1_M14    G1_M13    G1_M12    G1_M11    G1_M10    G1_M9    G1_M8    G1_M7    G1_M6    G1_M5    G1_M4    G1_M3    G1_M2    G1_M1    G1_M0																R    G1_M15    G1_M14    G1_M13    G1_M12    G1_M11    G1_M10    G1_M9    G1_M8    G1_M7    G1_M6    G1_M5    G1_M4    G1_M3    G1_M2    G1_M1    G1_M0			
W    1																W    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1			
PO Reset 1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1																PO Reset 1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1			

**Figure B-124. Secure Memory Access Groups Register, Group 1 - Format**

**Table B-110. Secure Memory Access Groups Register - Field Descriptions**

Field	Description
Group 2	
31-16 0	Reserved
15 G2_M15	Access Group 2 MID 15. Bit set to 1 indicates MID 15 is a member of Access Group 2
14 G2_M14	Access Group 2 MID 14. Bit set to 1 indicates MID 14 is a member of Access Group 2
13 G2_M13	Access Group 2 MID 13. Bit set to 1 indicates MID 13 is a member of Access Group 2
12 G2_M12	Access Group 2 MID 12. Bit set to 1 indicates MID 12 is a member of Access Group 2
11 G2_M11	Access Group 2 MID 11. Bit set to 1 indicates MID 11 is a member of Access Group 2
10 G2_M10	Access Group 2 MID 10. Bit set to 1 indicates MID 10 is a member of Access Group 2
9 G2_M9	Access Group 2 MID 9. Bit set to 1 indicates MID 9 is a member of Access Group 2
8 G2_M8	Access Group 2 MID 8. Bit set to 1 indicates MID 8 is a member of Access Group 2
7 G2_M7	Access Group 2 MID 7. Bit set to 1 indicates MID 7 is a member of Access Group 2
6 G2_M6	Access Group 2 MID 6. Bit set to 1 indicates MID 6 is a member of Access Group 2
5 G2_M5	Access Group 2 MID 5. Bit set to 1 indicates MID 5 is a member of Access Group 2
4 G2_M4	Access Group 2 MID 4. Bit set to 1 indicates MID 4 is a member of Access Group 2
3 G2_M3	Access Group 2 MID 3. Bit set to 1 indicates MID 3 is a member of Access Group 2
2 G2_M2	Access Group 2 MID 2. Bit set to 1 indicates MID 2 is a member of Access Group 2
1 G2_M1	Access Group 2 MID 1. Bit set to 1 indicates MID 1 is a member of Access Group 2
0 G2_M0	Access Group 2 MID 0. Bit set to 1 indicates MID 0 is a member of Access Group 2
Group 1	
31-16 0	Reserved
15 G1_M15	Access Group 1 MID 15. Bit set to 1 indicates MID 15 is a member of Access Group 1
14 G1_M14	Access Group 1 MID 14. Bit set to 1 indicates MID 14 is a member of Access Group 1
13 G1_M13	Access Group 1 MID 13. Bit set to 1 indicates MID 13 is a member of Access Group 1
12 G1_M12	Access Group 1 MID 12. Bit set to 1 indicates MID 12 is a member of Access Group 1
11 G1_M11	Access Group 1 MID 11. Bit set to 1 indicates MID 11 is a member of Access Group 1
10 G1_M10	Access Group 1 MID 10. Bit set to 1 indicates MID 10 is a member of Access Group 1
9 G1_M9	Access Group 1 MID 9. Bit set to 1 indicates MID 9 is a member of Access Group 1
8 G1_M8	Access Group 1 MID 8. Bit set to 1 indicates MID 8 is a member of Access Group 1
7 G1_M7	Access Group 1 MID 7. Bit set to 1 indicates MID 7 is a member of Access Group 1
6 G1_M6	Access Group 1 MID 6. Bit set to 1 indicates MID 6 is a member of Access Group 1
5 G1_M5	Access Group 1 MID 5. Bit set to 1 indicates MID 5 is a member of Access Group 1

Field	Description
4 G1_M4	Access Group 1 MID 4. Bit set to 1 indicates MID 4 is a member of Access Group 1
3 G1_M3	Access Group 1 MID 3. Bit set to 1 indicates MID 3 is a member of Access Group 1
2 G1_M2	Access Group 1 MID 2. Bit set to 1 indicates MID 2 is a member of Access Group 1
1 G1_M1	Access Group 1 MID 1. Bit set to 1 indicates MID 1 is a member of Access Group 1
0 G1_M0	Access Group 1 MID 0. Bit set to 1 indicates MID 0 is a member of Access Group 1

## NOTE

i.MX6 series users are not expected to require the endianness swapping capabilities of the module. This configurability exists to support other chips that may have mixed endian masters within a single chip.

## B.2.8 DECO and CCB Registers

This section describes the registers of the DECO and CCB.

### B.2.8.1 Class 1 Mode Register (CxC1MR)

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

#### B.2.8.1.1 Class 1 Mode Register Field Descriptions

The following figure defines the format of the Class 1 Mode Register. The field definitions appear in the figure below and the description and settings for the register are given in the following table.

Offset 0xBASE\_8004 (C0C1MR)

Access: Read/Write

(Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
(Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	ALG							
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	AAI						AS	ICV_TEST	ENC			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-125. Class 1 Mode Register - Format

Field	Description															
31-24	Reserved. Must be 0.															
23-16 ALG	Algorithm. This field specifies which algorithm is being selected. <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>10h</td> <td>AES</td> </tr> <tr> <td>20h</td> <td>DES</td> </tr> <tr> <td>0x21</td> <td>3DES</td> </tr> <tr> <td>30h</td> <td>ARC4</td> </tr> <tr> <td>50h</td> <td>RNG</td> </tr> <tr> <td>others</td> <td>Reserved</td> </tr> </tbody> </table>		Value	Description	10h	AES	20h	DES	0x21	3DES	30h	ARC4	50h	RNG	others	Reserved
Value	Description															
10h	AES															
20h	DES															
0x21	3DES															
30h	ARC4															
50h	RNG															
others	Reserved															
15-13	Reserved. Must be 0.															

## CAAM Register Descriptions

Field	Description																																																																																									
12-4 AAI	<p>Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. See also the section describing the appropriate CHA. Note that some algorithms do not require additional algorithm information and in those cases this field should be all 0s. Note that for RNG OPERATION commands the AAI field is interpreted as shown in the shaded SK, AI, PS, OBP, NZ and SH fields below.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="4">AAI Interpretation for AES Modes [For AES the MSB of AAI is the DK (Decrypt Key) bit.]</th> </tr> <tr> <th>Code 1</th> <th>Interpretation</th> <th>Code 1</th> <th>Interpretation</th> </tr> </thead> <tbody> <tr> <td>00h</td> <td>CTR (mod <math>2^{128}</math>)</td> <td>80h</td> <td>CCM<sup>2</sup></td> </tr> <tr> <td>10h</td> <td>CBC</td> <td>90h</td> <td>Reserved</td> </tr> <tr> <td>20h</td> <td>ECB</td> <td>A0h</td> <td>CBC_XCBC_MAC</td> </tr> <tr> <td>30h</td> <td>CFB</td> <td>B0h</td> <td>CTR_XCBC_MAC</td> </tr> <tr> <td>40h</td> <td>OFB</td> <td>C0h</td> <td>CBC_CMAC</td> </tr> <tr> <td>50h</td> <td>Reserved</td> <td>D0h</td> <td>CTR_CMAC_LTE</td> </tr> <tr> <td>60h</td> <td>CMAC</td> <td>E0h</td> <td>CTR_CMAC</td> </tr> <tr> <td>70h</td> <td>XCBC-MAC</td> <td></td> <td></td> </tr> <tr> <td colspan="4">           Setting the DK bit (i.e. ORing 10h0 with any AES code above) causes Key Register to be loaded with the AES <i>Decrypt</i> key, rather than the AES <i>Encrypt</i> key. .         </td><td></td></tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="4">AAI Interpretation for DES</th> </tr> <tr> <th>Code 1</th> <th>Interpretation</th> <th>Code 1</th> <th>Interpretation</th> </tr> </thead> <tbody> <tr> <td>10h</td> <td>CBC</td> <td>30h</td> <td>CFB</td> </tr> <tr> <td>20h</td> <td>ECB</td> <td>40h</td> <td>OFB</td> </tr> <tr> <td colspan="4">           80h ORed with any DES code above: Check odd parity         </td></tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="4">AAI Interpretation for RNG</th> </tr> <tr> <td></td> <td colspan="3"> <b>For RNG operations, see shaded rows below for the definitions of the 13 ls bits of the Class 1 Mode Register.</b> </td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> </thead> </table>	AAI Interpretation for AES Modes [For AES the MSB of AAI is the DK (Decrypt Key) bit.]				Code 1	Interpretation	Code 1	Interpretation	00h	CTR (mod $2^{128}$ )	80h	CCM <sup>2</sup>	10h	CBC	90h	Reserved	20h	ECB	A0h	CBC_XCBC_MAC	30h	CFB	B0h	CTR_XCBC_MAC	40h	OFB	C0h	CBC_CMAC	50h	Reserved	D0h	CTR_CMAC_LTE	60h	CMAC	E0h	CTR_CMAC	70h	XCBC-MAC			Setting the DK bit (i.e. ORing 10h0 with any AES code above) causes Key Register to be loaded with the AES <i>Decrypt</i> key, rather than the AES <i>Encrypt</i> key. .					AAI Interpretation for DES				Code 1	Interpretation	Code 1	Interpretation	10h	CBC	30h	CFB	20h	ECB	40h	OFB	80h ORed with any DES code above: Check odd parity				AAI Interpretation for RNG					<b>For RNG operations, see shaded rows below for the definitions of the 13 ls bits of the Class 1 Mode Register.</b>																		
AAI Interpretation for AES Modes [For AES the MSB of AAI is the DK (Decrypt Key) bit.]																																																																																										
Code 1	Interpretation	Code 1	Interpretation																																																																																							
00h	CTR (mod $2^{128}$ )	80h	CCM <sup>2</sup>																																																																																							
10h	CBC	90h	Reserved																																																																																							
20h	ECB	A0h	CBC_XCBC_MAC																																																																																							
30h	CFB	B0h	CTR_XCBC_MAC																																																																																							
40h	OFB	C0h	CBC_CMAC																																																																																							
50h	Reserved	D0h	CTR_CMAC_LTE																																																																																							
60h	CMAC	E0h	CTR_CMAC																																																																																							
70h	XCBC-MAC																																																																																									
Setting the DK bit (i.e. ORing 10h0 with any AES code above) causes Key Register to be loaded with the AES <i>Decrypt</i> key, rather than the AES <i>Encrypt</i> key. .																																																																																										
AAI Interpretation for DES																																																																																										
Code 1	Interpretation	Code 1	Interpretation																																																																																							
10h	CBC	30h	CFB																																																																																							
20h	ECB	40h	OFB																																																																																							
80h ORed with any DES code above: Check odd parity																																																																																										
AAI Interpretation for RNG																																																																																										
	<b>For RNG operations, see shaded rows below for the definitions of the 13 ls bits of the Class 1 Mode Register.</b>																																																																																									
3-2 AS	<p>Algorithm State. This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see the shaded AS field below.</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th colspan="5">AS Value</th> </tr> <tr> <th></th> <th>00</th> <th>01</th> <th>10</th> <th>11</th> </tr> </thead> <tbody> <tr> <td>Interpretation :</td> <td>Update</td> <td>Initialize</td> <td>Finalize</td> <td>Initialize/Finalize</td> </tr> </tbody> </table>				AS Value						00	01	10	11	Interpretation :	Update	Initialize	Finalize	Initialize/Finalize																																																																							
AS Value																																																																																										
	00	01	10	11																																																																																						
Interpretation :	Update	Initialize	Finalize	Initialize/Finalize																																																																																						

Field	Description
1 ICV_TES T	<p>ICV Checking / Test AES fault detection.  <i>(This is the definition of this bit for CHAs other than RNG. For the definition of this bit in RNG commands, see the shaded PR field below.)</i></p> <p>For algorithms other than AES ECB mode: ICV Checking      This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.</p> <p>0 - Don't compare      1 - Compare</p> <p>For AES ECB mode: Test AES fault detection      In AES ECB mode, this bit activates fault detection testing by injecting bit level errors into AES core logic as defined in the first 128 bits of the context.</p> <p>0 - Don't inject bit errors      1 - Inject bit errors</p>
0 ENC	<p>Encrypt/Decrypt.  <i>(This is the definition of this bit for CHAs other than RNG.. For the definition of this bit in RNG commands, see the shaded TST field below.)</i></p> <p>This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes.</p> <p>0 - Decrypt.      1 - Encrypt.</p>
The shaded rows below described how bits 12-0 are interpreted for RNG commands.	
12 SK	Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS≠00.
11 AI	Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the 256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Uninstantiate).
10 PS	Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional “entropy” input during instantiation. Note that the personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01.
9 OBP	Odd Byte Parity. For RNG OPERATION commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00.
8 NZB	NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00h, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00.

## CAAM Register Descriptions

Field	Description														
7-6	Reserved. For RNG commands these bits of the AAI field are reserved.														
5-4 SH	<p>State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented.</p> <p>00 - State Handle 0 01 - State Handle 1 10 - Reserved 11 - Reserved</p>														
3-2 AS	<p>Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown below:</p> <table border="1"> <thead> <tr> <th>AS Value</th> <th>State Handle is already instantiated</th> <th>State Handle is NOT already instantiated</th> </tr> </thead> <tbody> <tr> <td>00 Generate</td> <td>Generate random data per the mode in which the state handle was instantiated.</td> <td>Error</td> </tr> <tr> <td>01 Instantiate</td> <td>Error</td> <td>Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.</td> </tr> <tr> <td>10 Reseed</td> <td>Reseed the state handle.</td> <td>Error</td> </tr> </tbody> </table>			AS Value	State Handle is already instantiated	State Handle is NOT already instantiated	00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error	01 Instantiate	Error	Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.	10 Reseed	Reseed the state handle.	Error
AS Value	State Handle is already instantiated	State Handle is NOT already instantiated													
00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error													
01 Instantiate	Error	Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.													
10 Reseed	Reseed the state handle.	Error													

Field	Description		
1 PR	Prediction Resistance. For RNG OPERATION commands this bit is interpreted as:		
	AS Value	PR = 0	PR = 1
	00 Generate	Do NOT reseed prior to generating new random data	If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.
	01 Instantiate	Instantiate the state handle to NOT support prediction resistance	Instantiate the state handle to support prediction resistance
	10 Reseed	Reseed the state handle. PR bit is ignored.	Reseed the state handle. PR bit is ignored.
	11 Uninstantiate	Uninstantiate the state handle. PR bit is ignored	Uninstantiate the state handle. PR bit is ignored
0 TST	Test Mode Request. For RNG OPERATION commands this bit is interpreted as:		
	AS Value	TST = 0	TST = 1
	00 Generate	If the selected state handle is in nondeterministic mode, generate new random data.  If the selected state handle is in deterministic mode, generate a Test error. <sup>1</sup>	If the selected state handle is in deterministic mode, generate new random data.  If the selected state handle is in nondeterministic mode, generate a Test error..
	01 Instantiate	Instantiate the state handle in normal (nondeterministic) mode.	Instantiate the state handle in test (deterministic) mode.
	10 Reseed	If the selected state handle is in nondeterministic mode, reseed the state handle.  If the selected state handle is in deterministic mode, generate a Test error.	If the selected state handle is in nondeterministic mode, reseed the state handle.  If the selected state handle is in deterministic mode, generate a Test error.
	11 Uninstantiate	Uninstantiate the state handle. TST bit is ignored.	Uninstantiate the state handle. TST bit is ignored.

<sup>1</sup> There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests nondeterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

Some examples of how to build the Class 1 Mode Register are as follows:

**Table B-111. Class 1 Mode Register examples**

Crypto service performed	ALG Mnemonic	AAI Mnemonic	AS Mnemonic	ICV	Encrypt/Decrypt/Protect/Authenticate	ALGORITHM OPERATION Command	32-bit Value Loaded into C1 Mode Reg
AES GCM	AES	GCM	Init / Finalize	yes	Decrypt	0x8201090E	0x0001090E
AES Counter with mod=2 <sup>128</sup>	AES	CTR Modulus 2 <sup>128</sup>	--	no	Encrypt	0x82010001	0x00010001
Triple DES OFB mode with key parity	DES	OFB	--	no	Decrypt	0x82021400	0x00021400

### B.2.8.2 Class 1 Key Size Register (CxC1KSR)

The Class 1 Key Size Register is used to tell the Class 1 CHA the size of the key that was loaded into the Class 1 Key Register. The Class 1 Key Size Register must be written after the key is written into the Class 1 Key Register. Writing to the Class 1 Key Size Register will prevent the user from modifying the Class 1 Key Register. The bit assignments of this register appear in [Figure B-126](#). The Class 1 Key Size Register is automatically written by the KEY Command except in the following cases. When AFHA Sboxes are loaded the Class 1 Key Size Register is not loaded because no key size is required.

Offset 0xBASE\_800C (C0C1KSR)

Access: Read/Write

(Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
(Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-126. Class 1 Key Size Register - Format**

**Table B-112. Class 1 Key Size Register - Field Descriptions**

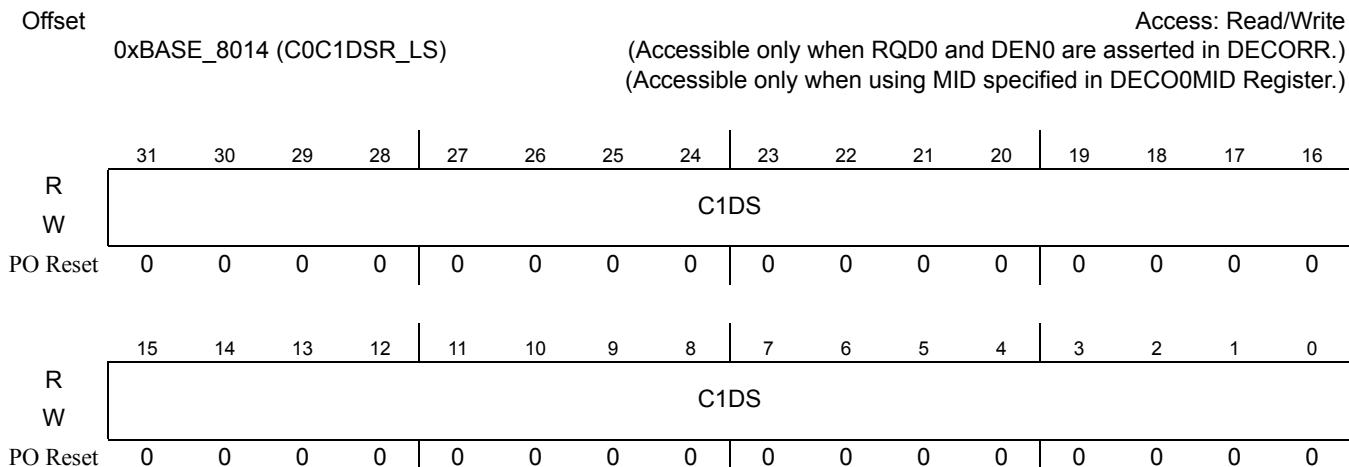
Field	Description
31-7	Reserved.
6-0 C1KS	Class 1 Key Size. This is the size of a Class 1 Key measured in bytes

### B.2.8.3 Class 1 Data Size Register (CxC1DSR)

The Class 1 Data Size Register is used to tell the Class 1 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C1CY and C1DS fields to form a data size that is measured in bits. Note that writing to the C1DS field in this register causes the written value to be added to the previous value in that field. That is, if the C1DS field currently has the value 14, writing 2 to the least-significant half of the Class 1 Data Size register (i.e. the C1DS field) will result in a value of 16 in the C1DS field. Although there is a C1CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C1CY and C1DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic iNformation FIFO entries are enabled. This register is cleared whenever a key is decrypted or encrypted. Since the Class 1 Data Size Registers hold more than 32 bits, they are accessed from the IP bus as two 32-bit registers. The bit assignments of this register appear in [Figure B-127](#).

Offset	0xBASE_8010 (C0C1DSR_MS)																Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
W	NUMBITS				0	0	0	0	0	0	0	0	0	0	0	0				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C1CY				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

**Figure B-127. Class 1 Data Size Register MS - Format**

**Figure B-128. Class 1 Data Size Register LS - Format****Table B-113. Class 1 Data Size Register - Field Descriptions**

Field	Description
Most-significant portion of register	
31-29 NUMBITS	Class 1 Data Size Number of bits. For bit-oriented operations, this value is appended to the C1CY and C1DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value ( C1CY    C1DS    NUMBITS ). Note that if NUMBITS is nonzero, C1DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value.
28-1	Reserved.
0 C1CY	Class 1 Data Size Carry. Although this field is not writable, it will be set if a write to C1DS causes a carry out of the msb of C1DS.
Least-significant portion of register	
31-0 C1DS	Class 1 Data Size. This is the number of whole bytes of data that will be consumed by the Class 1 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero.

### B.2.8.4 Class 1 ICV Size Register (CxC1ICVSR)

The Class 1 ICV Size Register indicates how much of the last block of ICV is valid when performing AES integrity check modes (e.g. AES-CMAC, AES-XCBC-MAC). Like the Class 1 Data Size register, the Class 1 ICV Size register is additive. That is, any value written to the C1ICVS field will be added to the previous value in the field. This register must be written prior to the corresponding word of data being consumed by AES. In practical terms, this means the register must be written either prior to the corresponding data being written to the Input Data FIFO or prior to the iNformation FIFO entry for this

data. The bit assignments of this register appear in Figure B-129. FIFO LOAD commands can automatically load it when ICV is loaded.

Offset																	Access: Read/Write
	0xBASE_801C (C0C1ICVSR)																(Accessible only when RQD0 and DEN0 are asserted in DECORR.)
	(Accessible only when using MID specified in DECO0MID Register.)																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C1ICVS
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B-129. Class 1 ICV Size Register - Format

Table B-114. Class 1 ICV Size Register - Field Descriptions

Field	Description
31-5	Reserved.
4-0 C1ICVS	Class 1 ICV Size, in Bytes.

### B.2.8.5 CHA Control Register (CCTRL)

The CHA Control Register is used to send control signals to the CHAs. This register is automatically written between Descriptors. Within a Descriptor, use the LOAD Command to reset blocks or unload memories.

## CAAM Register Descriptions

Offset 0xBASE\_8034 (C0CCTRL) Access: Read/Write  
 (Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
 (Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W				RSV	RSV	RSV	RNG	RSV	MD	RSV	RSV	RSV	RC4	DES	AES	ALL
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-130. CHA Control Register (CCTRL) - Format**

**Table B-115. CHA Control Register - Field Descriptions**

Field	Description
31-29	Reserved. To preserve software compatibility with other versions of CAAM, 0 should be written to all reserved bits (shaded or marked "RSV")
28	Reserved.
27-13	Reserved.
12	Reserved.
11	Reserved..
10	Reserved.
9 RNG	Reset Random Number Generator. Writing a 1 to this bit resets the Random Number Generator. 0 Do Not Reset 1 Reset Random Number Generator Block.
8	Reserved.
7 MD	Reset Hashing Block. Writing a 1 to this bit resets the MDHA block. 0 Do Not Reset 1 Reset Hashing Block
6	Reserved.
5	Reserved.
4	Reserved.

Field	Description
3 RC4	Reset RC4. Writing a 1 to this bit resets the ARC4 block. 0 Do Not Reset 1 Reset RC4
2 DES	Reset DES. Writing a 1 to this bit resets the DES block. 0 Do Not Reset 1 Reset DES
1 AES	Reset AES. Writing a 1 to this bit resets the AESA. 0 Do Not Reset 1 Reset AES
0 ALL	Reset All Internal CHAs. Writing to this bit resets all CHAs in use by this CCB. 0 Do Not Reset 1 Reset all CHAs in use by this CCB.

### B.2.8.6 IRQ Control Register (CxCIRQ)

The IRQ Control Register shows the status of all CCB “done” interrupts and “error” interrupts and provides controls for clearing these interrupts.

Offset	0xBASE_803C (C0C1CIRQ)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)													
R	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16											
W	0 0 0 0	0 0 RNEI 0	MEI 0 0 0	RCEI DEI AEI 0											
PO Reset	0 0 0 0	RSV RSV w1c RSV	w1c RSV RSV RSV	w1c w1c w1c w1c											
R	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0											
W	0 0 0 0	0 0 RNDI 0	MDI 0 0 0	RCDI DDI ADI 0											
PO Reset	0 0 0 0	RSV RSV w1c RSV	w1c RSV RSV RSV	w1c w1c w1c w1c											

**Figure B-131. Interrupt Control Register (CIRQ) - Format**

**Table B-116. Interrupt Control Register - Field Descriptions**

Field	Description
31-29	Reserved. To preserve software compatibility with other versions of CAAM, 0 should be written to all reserved bits (shaded or marked “RSV”)
28	Reserved

## CAAM Register Descriptions

Field	Description
27	Reserved
26	Reserved.
25 RNEI	RNG error interrupt asserted. Read: 0 No Error Write: 0 No change Read:1 RNG Error Interrupt Asserted Write:1 Clear the RNG Error Interrupt
24	Reserved.
23 MEI	MDHA (hashing) error interrupt asserted. Read: 0 No Error Write: 0 No change Read: 1 MDHA Error Interrupt Asserted Write: 1 Clear the MDHA Error Interrupt
22	Reserved
21	Reserved.
20	Reserved.
19 RCEI	ARC4 error interrupt asserted. Read: 0 No Error Write: 0 No change Read:1 ARC4 Error Interrupt Asserted Write:1 Clear the ARC4 Error Interrupt
18 DEI	DESA error interrupt asserted. Read: 0 No Error Write: 0 No change Read: 1 DESA Error Interrupt Asserted Write: 1 Clear the DESA Error Interrupt
17 AEI	AESA error interrupt asserted. Read: 0 No Error Write: 0 No change Read: 1 AESA Error Interrupt Asserted Write: 1 Clear the AESA Error Interrupt
16-13	Reserved.
12	Reserved
11	Reserved
10	Reserved.

Field	Description
9 RNDI	RNG done interrupt. Read: 0 No Done Interrupt Write: 0 No change Read: 1 RNG Done Interrupt Asserted Write: 1 Clear the RNG Done Interrupt
8	Reserved.
7 MDI	MDHA (hashing) done interrupt. Read: 0 No Done Interrupt Write: 0 No change Read: 1 MDHA Done Interrupt Asserted Write: 1 Clear the MDHA Done Interrupt
6	Reserved.
5	Reserved.
4	Reserved.
3 RCDI	ARC4 done interrupt. Read: 0 No Done Interrupt Write: 0 No change Read: 1 ARC4 Done Interrupt Asserted Write: 1 Clear the ARC4 Done Interrupt
2 DDI	DESA done interrupt. Read: 0 No Done Interrupt Write: 0 No change Read: 1 DESA Done Interrupt Asserted Write: 1 Clear the DESA Done Interrupt
1 ADI	AESA done interrupt. Read: 0 No Done Interrupt Write: 0 No change Read: 1 AESA Done Interrupt Asserted Write: 1 Clear the AESA Done Interrupt
0	Reserved.

### B.2.8.7 Clear Written Register (CxCWR)

The Clear Written Register is used to clear many of the internal registers. This register is automatically written, if necessary, by DECO between Shared Descriptors. All fields of this register are self-clearing.

Offset 0xBASE\_8044 (C0CWR) Access: Read/Write  
 (Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
 (Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CIF	COF	CIRST	C2RST	C1D	C2D			C2K	C2C			C2DS		C2M	
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W									C1K	C1C			C1ICV	C1DS		C1M
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-132. Clear Written Register (CWR) - Format****Table B-117. Clear Written Register - Field Descriptions**

Field	Description
31 CIF	Clear Input FIFO (and NFIFO). Writing a 1 to this bit causes the Input Data FIFO and iNformation FIFO to be cleared.
30 COF	Clear Output FIFO. Writing a 1 to this bit causes the Output FIFO to be cleared.
29 CIRST	Reset Class 1 CHA. Writing a 1 to this bit causes a reset to any Class 1 CHA that is currently selected by this DECO.
28 C2RST	Reset Class 2 CHA. Writing a 1 to this bit causes a reset to any Class 2 CHA that is currently selected by this DECO.
27 C1D	Clear Class 1 Done Interrupt. Writing a 1 to this bit clears the Class 1 done interrupt.
26 C2D	Clear Class 1 Done Interrupt. Writing a 1 to this bit clears the Class 2 done interrupt.
25-23	Reserved.
22 C2K	Clear the Class 2 Key Register. Writing a one to this bit causes the Class 2 Key and Key Size Registers to be cleared.
21 C2C	Clear the Class 2 Context Register. Writing a one to this bit causes the Class 2 Context Register to be cleared.
20-19	Reserved.
18 C2DS	Clear the Class 2 Data Size Registers. Writing a one to this bit causes the Class 2 Data Size and ICV Size Registers to be cleared.
17	Reserved.
16 C2M	Clear the Class 2 Mode Register. Writing a one to this bit causes the Class 2 Mode Register to be cleared.
15-7	Reserved.

Field	Description
6 C1K	Clear the Class 1 Key Register. Writing a one to this bit causes the Class 1 Key and Key Size Registers to be cleared.
5 C1C	Clear the Class 1 Context Register. Writing a one to this bit causes the Class 1 Context Register to be cleared.
4	Reserved.
3 C1ICV	Clear the Class 1 ICV Size Register. Writing a one to this bit causes the Class 1 ICV Size Register to be cleared.
2 C1DS	Clear the Class 1 Data Size Register. Writing a one to this bit causes the Class 1 Data Size Register to be cleared. This clears AAD Size as well.
1	Reserved.
0 C1M	Clear the Class 1 Mode Register. Writing a one to this bit causes the Class 1 Mode Register to be cleared.

### B.2.8.8 CCB Status and Error Register (CxCSTA)

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CxCSTA are accessed as two 32-bit registers.

Offset	0xBASE_8048 (C0CSTA_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	CL2 0 0 0 0   0 0 0 0   0 0 0 0   ERRID2 0 0 0 0
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	CL1 0 0 0 0   0 0 0 0   0 0 0 0   ERRID1 0 0 0 0
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-133. CCB Status and Error Register MS (CxCSTA MS) - Format**

## CAAM Register Descriptions

Offset 0xBASE\_804C (C0CSTA\_LS) Access: Read/Write  
 (Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
 (Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	PIZ	GCD	PRM	0	0	0	0	0	0	SEI	PEI	0	0	SDI	PDI
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	RNB	0	MB	PB	0	0	RCB	DB	AB	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-134. CCB Status and Error Register\_LS (CxCSTA\_LS) - Format**

**Table B-118. CCB Status and Error Register - Field Descriptions**

Field	Description																					
Most significant portion of register.																						
31-28 CL2	Class 2 Algorithms. The Class 2 Algorithms bits indicate which algorithm is asserting an error.																					
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0100</td> <td>MD5, SHA-1, SHA-224, SHA-256</td> </tr> <tr> <td>Others</td> <td>Reserved</td> </tr> </tbody> </table>		Value	Description	0100	MD5, SHA-1, SHA-224, SHA-256	Others	Reserved														
Value	Description																					
0100	MD5, SHA-1, SHA-224, SHA-256																					
Others	Reserved																					
27-20	Reserved.																					
19-16 ERRID2	Error ID 2. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL2 field.																					
	<table border="1"> <thead> <tr> <th>ERRID2</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0001</td> <td>Mode Error</td> </tr> <tr> <td>0010</td> <td>Data Size Error</td> </tr> <tr> <td>0011</td> <td>Key Size Error</td> </tr> <tr> <td>0110</td> <td>Data Arrived out of Sequence Error</td> </tr> <tr> <td>1010</td> <td>ICV Check Failed</td> </tr> <tr> <td>1011</td> <td>Internal Hardware Failure</td> </tr> <tr> <td>1110</td> <td>Invalid CHA combination was selected.</td> </tr> <tr> <td>1111</td> <td>Invalid CHA Selected</td> </tr> <tr> <td>Others</td> <td>Reserved</td> </tr> </tbody> </table>		ERRID2	Description	0001	Mode Error	0010	Data Size Error	0011	Key Size Error	0110	Data Arrived out of Sequence Error	1010	ICV Check Failed	1011	Internal Hardware Failure	1110	Invalid CHA combination was selected.	1111	Invalid CHA Selected	Others	Reserved
ERRID2	Description																					
0001	Mode Error																					
0010	Data Size Error																					
0011	Key Size Error																					
0110	Data Arrived out of Sequence Error																					
1010	ICV Check Failed																					
1011	Internal Hardware Failure																					
1110	Invalid CHA combination was selected.																					
1111	Invalid CHA Selected																					
Others	Reserved																					

Field	Description																											
15-12 CL1	Class 1 algorithms. The Class 1 algorithms field indicates which algorithm is asserting an error.																											
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0001</td><td>AES</td></tr> <tr> <td>0010</td><td>DES</td></tr> <tr> <td>0011</td><td>ARC4</td></tr> <tr> <td>0101</td><td>RNG</td></tr> <tr> <td>Others</td><td>Reserved</td></tr> </tbody> </table>		Value	Description	0001	AES	0010	DES	0011	ARC4	0101	RNG	Others	Reserved														
Value	Description																											
0001	AES																											
0010	DES																											
0011	ARC4																											
0101	RNG																											
Others	Reserved																											
11-4	Reserved																											
3-0 ERRID1	Error ID 1. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL1 field.																											
	<table border="1"> <thead> <tr> <th>ERRID1</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0001</td><td>Mode Error</td></tr> <tr> <td>0010</td><td>Data Size Error</td></tr> <tr> <td>0011</td><td>Key Size Error</td></tr> <tr> <td>0110</td><td>Data Arrived out of Sequence Error</td></tr> <tr> <td>1001</td><td>DES Key Parity Error</td></tr> <tr> <td>1010</td><td>ICV Check Failed</td></tr> <tr> <td>1011</td><td>Internal Hardware Failure</td></tr> <tr> <td>1100</td><td>CCM AAD Size Error (either 1. AAD flag in B0 =1 and no AAD type provided, 2. AAD flag in B0 =0 and AAD povidied, or 3. AAD flag in B0 =1 and not enough AAD provided - expecting more based on AAD size.)</td></tr> <tr> <td>1101</td><td>Class 1 CHA is not reset</td></tr> <tr> <td>1110</td><td>Invalid CHA combination was selected.</td></tr> <tr> <td>1111</td><td>Invalid CHA Selected</td></tr> <tr> <td>Others</td><td>Reserved</td></tr> </tbody> </table>		ERRID1	Description	0001	Mode Error	0010	Data Size Error	0011	Key Size Error	0110	Data Arrived out of Sequence Error	1001	DES Key Parity Error	1010	ICV Check Failed	1011	Internal Hardware Failure	1100	CCM AAD Size Error (either 1. AAD flag in B0 =1 and no AAD type provided, 2. AAD flag in B0 =0 and AAD povidied, or 3. AAD flag in B0 =1 and not enough AAD provided - expecting more based on AAD size.)	1101	Class 1 CHA is not reset	1110	Invalid CHA combination was selected.	1111	Invalid CHA Selected	Others	Reserved
ERRID1	Description																											
0001	Mode Error																											
0010	Data Size Error																											
0011	Key Size Error																											
0110	Data Arrived out of Sequence Error																											
1001	DES Key Parity Error																											
1010	ICV Check Failed																											
1011	Internal Hardware Failure																											
1100	CCM AAD Size Error (either 1. AAD flag in B0 =1 and no AAD type provided, 2. AAD flag in B0 =0 and AAD povidied, or 3. AAD flag in B0 =1 and not enough AAD provided - expecting more based on AAD size.)																											
1101	Class 1 CHA is not reset																											
1110	Invalid CHA combination was selected.																											
1111	Invalid CHA Selected																											
Others	Reserved																											
Least significant portion of register.																												
31-22	Reserved.																											
21 SEI	Class 2 Error Interrupt. The Class 2 Error Interrupt has been asserted. 0 No Error. 1 Error Interrupt.																											
20 PEI	Class 1 Error Interrupt. The Class 1 Error Interrupt has been asserted. 0 Not Error. 1 Error Interrupt.																											
19-18	Reserved.																											

## CAAM Register Descriptions

Field	Description
17 SDI	Class 2 Done Interrupt. The Class 2 Done Interrupt has been asserted. 0 Not Done. 1 Done Interrupt.
16 PDI	Class 1 Done Interrupt. The Class 1 Done Interrupt has been asserted. 0 Not Done. 1 Done Interrupt.
15-13	Reserved.
12	Reserved.
11	Reserved.
10	Reserved.
9 RNB	RNG Block Busy. This bit indicates that the RNG block is busy. The CHA can either be busy processing data or resetting. 0 RNG Idle 1 RNG Busy.
8	Reserved.
7 MB	Hashing Block Busy. This bit indicates that the Hashing block is busy. The CHA can either be busy processing data or resetting. 0 Hash Block Idle 1 Hash Block Busy.
6	Reserved.
5	Reserved.
4	Reserved.
3 RCB	ARC4 Block Busy. This bit indicates that the ARC4 block is busy. The CHA can either be busy processing data or resetting. 0 ARC4 Block Idle 1 ARC4 Block Busy.
2 DB	DES Block Busy. This bit indicates that the DES block is busy. The CHA can either be busy processing data or resetting. 0 DES Block Idle 1 DES Block Busy.
1 AB	AES Block Busy. This bit indicates that the AES block is busy. The CHA can either be busy processing data or resetting. 0 AES Block Idle 1 AES Block Busy.
0	Reserved.

### B.2.8.9 AAD Size Register (CxAADSZR)

The AAD Size Register is used by AESA to determine how much of the last block of AAD is valid. Like the Class 1 Data Size Register, writing to this register causes the written value to be added to the previous value in the register. The bit assignments of this register appear in [Figure B-135](#). The register is automatically written by FIFO LOAD commands.

																Access: Read/Write				
Offset	0xBASE_805C (C0AADSZR)																(Accessible only when RQD0 and DEN0 are asserted in DECORR.)			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	AASZ			
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
W																				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

**Figure B-135. AAD Size Register (CxAADSZR) - Format**

**Table B-119. AAD Size Register - Field Descriptions**

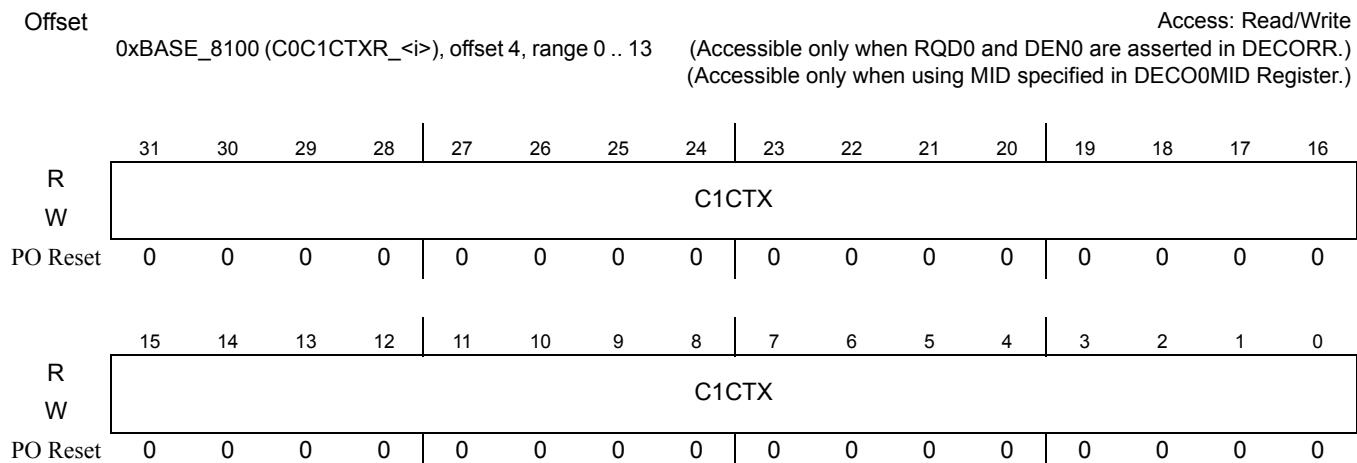
Field	Description
31-4	Reserved.
3-0 AASZ	AAD size in Bytes, mod 16.

### B.2.8.10 Class 1 Context Register (CxC1CTXR)

The Class 1 Context Register holds the context for the Class 1 CHAs. This register is 448 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Context Register is accessible only as full-word reads or writes to fourteen 32-bit registers. The MSB is in address 0x8100. This register is cleared automatically when a Black Key is being encrypted or decrypted using AES-CCM.

Note that some commands must block until a previous load to the Class 1 Context Register has completed. Loading the Class 1 Context Register, whether via the KEY Command, LOAD Command or MOVE Command, sets an internal blocking flag until the Class 1 Context Register load has completed.

The bit assignments of this register are dependent on the algorithm, and in some cases the mode of that algorithm. See the appropriate section for the Context Register format used for that algorithm.



**Figure B-136. Class 1 Context Register - Format**

### B.2.8.11 Class 1 Key Register (CxC1KEYR)

The Class 1 Key Register holds the left-aligned key for the Class 1 CHAs. The Class 1 Key Register is never readable directly. The FIFO STORE Command may be used to store an encrypted copy of this key. The encrypted key may be decrypted using the KEY Command as long as the KEK has not been changed (as a consequence of a security violation or a POR). This register is 256 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Key Register is accessible only as full-word reads or writes to eight 32-bit registers. The MSB is in address ?200h. This register is automatically written by KEY Commands. The recommended practice is to write the Class 1 Key Register prior to writing any of the other Class 1 registers. This register is cleared when any key is encrypted or decrypted. If a Class 1 key is being decrypted, the decrypted key is placed in this register automatically. Once the Class 1 Key Size Register is written, the Class 1 Key Register can no longer be modified by software or the descriptor until the key and key size are cleared.

Offset	0xBASE_8200 (C0C1KEYR_<i>), offset 4, range 0 .. 7	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16  C1KEYR  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0  C1KEYR  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

Figure B-137. Class 1 Key Register - Format

### B.2.8.12 Class 2 Mode Register (CxC2MR)

The Class 2 Mode Register is used to tell the Class 2 CHA which operation is being requested. The interpretation of this register is unique for each CHA. The Class 2 Mode Register is automatically written by the OPERATION Command. This register is automatically cleared when the signature over a trusted descriptor is checked or a trusted descriptor is re-signed. The bit assignments of this register appear in the following figure and the description and settings for the register are given in the following table.

Offset	0xBASE_8404 (C0C2MR)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16  ALG  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0  AAI  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	AS   ICV   AP

Figure B-138. Class 2 Mode Register - Format

Field	Description													
31-24	Reserved. Must be 0.													
23-16 ALG	Algorithm. This field specifies which algorithm is being requested to execute a job. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>40h</td> <td>MD5</td> </tr> <tr> <td>0x41</td> <td>SHA-1</td> </tr> <tr> <td>0x42</td> <td>SHA-224</td> </tr> <tr> <td>0x43</td> <td>SHA-256</td> </tr> <tr> <td>Others</td> <td>Reserved.</td> </tr> </tbody> </table>		Value	Description	40h	MD5	0x41	SHA-1	0x42	SHA-224	0x43	SHA-256	Others	Reserved.
Value	Description													
40h	MD5													
0x41	SHA-1													
0x42	SHA-224													
0x43	SHA-256													
Others	Reserved.													
15-13	Reserved. Must be 0.													
12-4 AAI	Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. A detailed list of additional modes can be found below. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Value</th> <th>Description</th> <th>Valid with ALG</th> </tr> </thead> <tbody> <tr> <td>0x001</td> <td>HMAC</td> <td>MD5, SHA-*</td> </tr> <tr> <td>0x004</td> <td>IPAD OPAD Generation</td> <td>MD5, SHA-*</td> </tr> <tr> <td>Others</td> <td>Reserved.</td> <td></td> </tr> </tbody> </table>		Value	Description	Valid with ALG	0x001	HMAC	MD5, SHA-*	0x004	IPAD OPAD Generation	MD5, SHA-*	Others	Reserved.	
Value	Description	Valid with ALG												
0x001	HMAC	MD5, SHA-*												
0x004	IPAD OPAD Generation	MD5, SHA-*												
Others	Reserved.													
3-2 AS	Algorithm State. This field defines the state of the algorithm that is being executed. Not every algorithm uses this field. Check the individual algorithm sections to see if this field is used. 00 Update. 01 Initialize. 10 Finalize. 11 Initialize/Finalize.													
1 ICV	ICV Checking. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.													
0 AP	Authenticate / Protect. 0 = Authenticate 1 = Protect.													

### B.2.8.13 Class 2 Key Size Register (CxC2KSR)

The Class 2 Key Size Register is used to tell the Class 2 CHA the size of the key that was loaded into the Class 2 Key Register. The Class 2 Key Size Register must be written after the key is written into the Class 2 Key Register. Writing to the Class 2 Key Size Register will prevent the user from modifying the Class 2 Key Register. The bit assignments of this register appear in [Figure B-139](#). The Class 2 Key Size Register is automatically written by the Key Command. This register is cleared when trusted descriptors are checked or re-signed.

Offset	0xBASE_840C (C0C2KSR)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	C2KS
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-139. Class 2 Key Size Register - Format****Table B-120. Class 2 Key Size Register - Field Descriptions**

Field	Field Description
31-8	Reserved.
7-0 C2KS	Class 2 key size in bytes.

### B.2.8.14 Class 2 Data Size Register (CxC2DSR)

The Class 2 Data Size Register is used to tell the Class 2 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C2CY and C2DS fields to form a data size that is measured in bits. Note that writing to the C2DS field in this register causes the written value to be added to the previous value in that field. That is, if the C2DS field currently has the value 14, writing 2 to the least-significant half of the Class 2 Data Size register (i.e. the C2DS field) will result in a value of 16 in the C2DS field. Although there is a C2CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C2CY and C2DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic iNformation FIFO entries are enabled. This register is reset when checking the signature over, or re-signing, Trusted Descriptors. Since the Class 2 Data Size Register holds more than 32 bits, it is accessed from the IP bus as two 32-bit registers. The bit assignments of this register appear in [Figure B-141](#).

## CAAM Register Descriptions

Offset	0xBASE_8410 (C0C2DSR_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	NUMBITS	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	C2CY
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-140. Class 2 Data Size Register\_MS - Format**

Offset	0xBASE_8414 (C0C2DSR_LS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	
W	C2DS	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	
W	C2DS	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-141. Class 2 Data Size Register\_LS - Format**

**Table B-121. Class 2 Data Size Register - Field Descriptions**

Field	Description
Most-significant portion of register	
31-29 NUMBITS	Class 2 Data Size Number of bits. For bit-oriented operations, this value is appended to the C2CY and C2DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value ( C2CY    C2DS    NUMBITS ). Note that if NUMBITS is nonzero, C2DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value.
28-1	Reserved.
0 C2CY	Class 2 Data Size Carry. Although this field is not writable, it will be set if a write to C2DS causes a carry out of the msb of C2DS.

Field	Description
Least-significant portion of register	
31-0 C2DS	Class 2 Data Size in Bytes. This is the number of whole bytes of data that will be consumed by the Class 2 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero.

### B.2.8.15 Class 2 ICV Size Register (Cx2ICVSZR)

The Class 2 ICV Size Register indicates how much of the last block of ICV is valid when performing MDHA integrity check operations (e.g. SHA-1, SHA-224, SHA-256 and MD5). Writing to this register causes the written value to be added to the previous value in the register. The bit assignments of these registers appear in [Figure B-142](#). This register is automatically written by FIFO LOAD commands. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

Offset	0xBASE_841C (C0C2ICVSZR)																Access: Read/Write															
	(Accessible only when RQD0 and DEN0 are asserted in DECORR.)																(Accessible only when using MID specified in DECO0MID Register.)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ICVSZ															
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																

**Figure B-142. Class 2 ICV Size Register (Cx2ICVSZR) - Format**

**Table B-122. Class 2 ICV Size Register - Field Descriptions**

Field	Description
31-3	Reserved.
2-0 ICVSZ	Class 2 ICV size (mod 8) in bytes. Writing 0h to this field will be interpreted as an ICV size of 8 bytes.

### B.2.8.16 Class 2 Context Register (Cx2CTXR)

The Class 2 Context Register holds the context for the Class 2 CHAs. This register is 320 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Context Register is accessible only as

full-word reads or writes to ten 32-bit registers. The MSB is in address 0x?500. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

The bit assignments for this register are dependent on the algorithm. See the appropriate section for the Context Register format used by that algorithm.

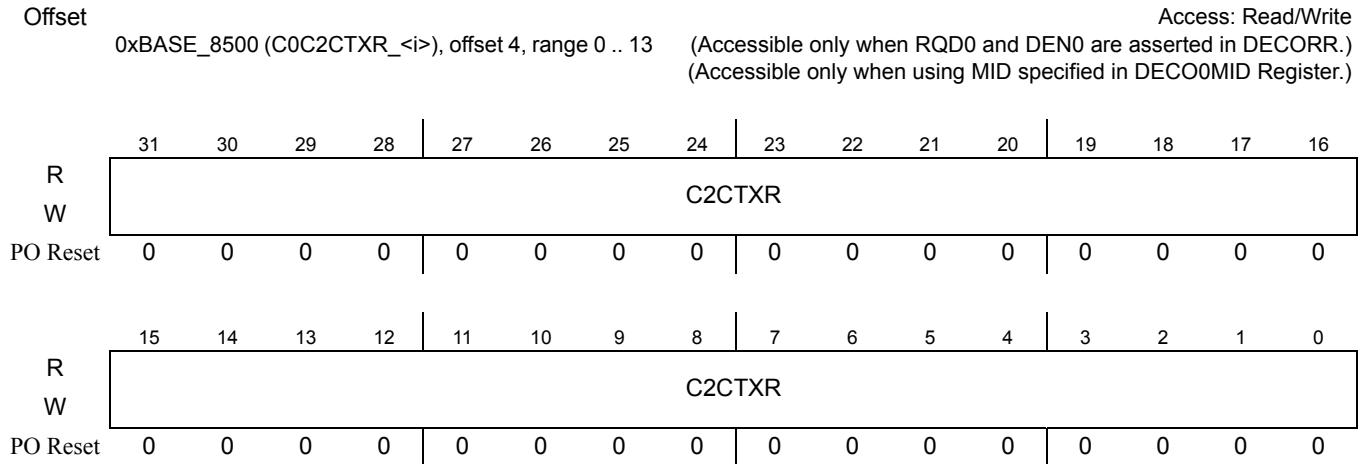


Figure B-143. Class 2 Context Register - Format

### B.2.8.17 Class 2 Key Register (Cx2KEYR)

The Class 2 Key Register holds the key for the Class 2 CHAs. For non-split keys, the key is left-aligned in the C2KEYR. Split keys are the IPAD and OPAD for certain MDHA operations. Note that the size of a split key is the sum of the sizes of the IPAD and the OPAD. The IPAD is left-aligned within the C2KEYR, and the OPAD is left-aligned starting at the mid-point of the C2KEYR. This register is 512 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Key Register is accessible only as full-word reads or writes to sixteen 32-bit registers. The MSB is in address 0x?600. This register is automatically written by KEY commands. The recommended practice is to write the Class 2 Key Register prior to writing any of the other Class 2 registers. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors. The format of the C2KEYR is shown in Figure B-144.

Offset	0xBASE_8600 (C0C2KEYR_<i>), offset 4, range 0 .. 13	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16  C2KR  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0  C2KR  PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-144. Class 2 Key Register - Format**

### B.2.8.18 iNformation FIFO (CxNFIFO)

The iNformation FIFO (Input Information FIFO) is used to control the movement of data from any of four sources to any of the three alignment blocks (see the “Cryptographic Acceleration and Assurance Module (CAAM)” chapter). The four sources are the Input Data FIFO, the Output Data FIFO, the CCB Padding Block and the Auxiliary Data FIFO. Note that the only way to get data out of any of these sources other than via the Output Data FIFO is to use an iNformation FIFO entry.

The depth of the iNformation FIFO is four entries. During normal operation, CAAM will not cause the iNformation FIFO to overflow. Care must be taken to avoid overflowing the iNformation FIFO when writing to it directly as this can cause CAAM to hang. This register can be automatically written by the FIFO LOAD and MOVE commands. (If data is written to the Input Data FIFO with the LOAD Command, or some other way, the user is responsible for writing to the iNformation FIFO.)

A single address is used to write to the iNformation FIFO. The format of non-padding iNformation FIFO entries (STYPE <> 10) is shown in Figure B-145. The format of padding iNformation FIFO entries (STYPE = 10) is shown in Figure B-146.

## CAAM Register Descriptions

Offset 0xBASE\_87D0 (C0NFIFO) Access: Read/Write  
 (Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
 (Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	DEST	LC2	LC1	FC2	FC1	STYPE	DTYPE				BND	PTYPE				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	OC	AST			DL											
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-145. iNformation FIFO (NFIFO) - Format when STYPE is not 10

Offset 0xBASE\_87D0 (C0NFIFO) Access: Read/Write  
 (Accessible only when RQD0 and DEN0 are asserted in DECORR.)  
 (Accessible only when using MID specified in DECO0MID Register.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	DEST	LC2	LC1	FC2	FC1	STYPE	DTYPE				BND	PTYPE				
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	BM	PS						PL			
R	PR				0	0	0	0	0	0	0	0	0	0	0	0

Figure B-146. iNformation FIFO (NFIFO) - Format when STYPE = 10

Field	Description
31-30 DEST	Destination. This specifies if the current entry defines data for the Class 1 CHA and/or Class 2 CHA. It can also be used to remove data from the FIFOs that are not needed. 00 DECO Alignment Block. If DTYPEn is Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPEn other than Eh (drop) or Fh (message) is used with the DECO Alignment Block destination. 01 Class 1. 10 Class 2. 11 Both Class 1 and Class 2.
29 LC2	Last Class 2. This bit should be set when the data defined in the current iNformation FIFO entry is the last data going to the CHA or the last data prior to receiving ICV data going to the Class 2 CHA, as well as following the ICV data. When LC2 =1 the alignment block will be emptied as well.
28 LC1	Last Class 1. This bit should be set when the data defined in the current iNformation FIFO entry is the last data for the Class 1 CHA. When LC1 =1 a flush will be done and the alignment block will be emptied as well.

Field	Description																											
27 FC2	<p>Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted.</p> <p>This bit can only be set via a LOAD Command and is only to be used when a MOVE from the Class 2 Alignment Block is to be done and the MOVE Command was executed when automatic information FIFO entries were disabled. In such cases, setting the LC2 bit could result in unpredictable behavior and the FC2 bit should be used.</p>																											
26 FC1	Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.																											
25-24 STYPE	<p>Source Type. This field defines the source of the data for the Alignment Block(s). There are two interpretations of the STYPE field, depending on the setting of the AST bit:</p> <table style="margin-left: 40px;"> <tr> <td>AST=0</td> <td>AST=1</td> </tr> <tr> <td>STYPE = 00 : Input Data FIFO</td> <td>STYPE = 00 : Auxiliary Data FIFO**</td> </tr> <tr> <td>STYPE = 01 : Output Data FIFO</td> <td>STYPE = 01 : reserved</td> </tr> <tr> <td>STYPE = 10 : Padding Block</td> <td>STYPE = 10 : reserved</td> </tr> <tr> <td>STYPE = 11 : Out snooping*</td> <td>STYPE = 11 : reserved</td> </tr> </table> <p>* When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.</p> <p>** The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=0x78 or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO must be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO will hang.</p>	AST=0	AST=1	STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO**	STYPE = 01 : Output Data FIFO	STYPE = 01 : reserved	STYPE = 10 : Padding Block	STYPE = 10 : reserved	STYPE = 11 : Out snooping*	STYPE = 11 : reserved																	
AST=0	AST=1																											
STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO**																											
STYPE = 01 : Output Data FIFO	STYPE = 01 : reserved																											
STYPE = 10 : Padding Block	STYPE = 10 : reserved																											
STYPE = 11 : Out snooping*	STYPE = 11 : reserved																											
23-20 DTYPE	<p>Data Type. This field defines the type of data that is going through the Input Data FIFO. This is used by the CHA to determine what type of processing needs to be done on the data.</p> <table style="margin-left: 100px;"> <tr> <td>DTYPE</td> <td></td> <td></td> </tr> <tr> <td>0h</td> <td></td> <td>S-Box (AFHA)</td> </tr> <tr> <td>1h</td> <td></td> <td>AAD (AES A ES GCM)</td> </tr> <tr> <td>2h</td> <td></td> <td>IV for (AES A ES GCM)</td> </tr> <tr> <td>3h</td> <td></td> <td>SAD (AES A)</td> </tr> <tr> <td>Ah</td> <td></td> <td>ICV</td> </tr> <tr> <td>Eh</td> <td></td> <td>DECO Ignore (i.e. Skip)</td> </tr> <tr> <td>Fh</td> <td></td> <td>Message Data</td> </tr> <tr> <td colspan="3">Other values are reserved.</td></tr> </table>	DTYPE			0h		S-Box (AFHA)	1h		AAD (AES A ES GCM)	2h		IV for (AES A ES GCM)	3h		SAD (AES A)	Ah		ICV	Eh		DECO Ignore (i.e. Skip)	Fh		Message Data	Other values are reserved.		
DTYPE																												
0h		S-Box (AFHA)																										
1h		AAD (AES A ES GCM)																										
2h		IV for (AES A ES GCM)																										
3h		SAD (AES A)																										
Ah		ICV																										
Eh		DECO Ignore (i.e. Skip)																										
Fh		Message Data																										
Other values are reserved.																												
19 BND	Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE ≠ Padding Block.																											

Field	Description										
18-16 PTYPE	Pad Type. This field defines the type of padding that should be performed when the STYPE = Padding Block. This field is ignored if BND = 0 or STYPE ≠ Padding Block. 000All Zero. 001Random with nonzero bytes. 010Incremented (Starting with 01h). 011Random. 100All Zero with last byte containing the number of 0 bytes. 101Random with nonzero bytes with last byte 0. 110N bytes of padding all containing the number N -1. 111Random with nonzero bytes with last byte N.										
Format of bits 15-0 when STYPE is not 10 (non-padding entries)											
15 OC	OFIFO Continuation - This bit causes the final word to not be popped from the Output Data FIFO.										
14 AST	Additional Source Types. This bit selects between two meanings of the STYPE field:  <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">AST=0</td> <td style="text-align: center;">AST=1</td> </tr> <tr> <td>STYPE = 00 : Input Data FIFO</td> <td>STYPE = 00 : Auxiliary Data FIFO</td> </tr> <tr> <td>STYPE = 01 : Output Data FIFO</td> <td>STYPE = 01 : reserved</td> </tr> <tr> <td>STYPE = 10 : Padding Block</td> <td>STYPE = 10 : reserved</td> </tr> <tr> <td>STYPE = 11 : Out snooping*</td> <td>STYPE = 11 : reserved</td> </tr> </table> * When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.	AST=0	AST=1	STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO	STYPE = 01 : Output Data FIFO	STYPE = 01 : reserved	STYPE = 10 : Padding Block	STYPE = 10 : reserved	STYPE = 11 : Out snooping*	STYPE = 11 : reserved
AST=0	AST=1										
STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO										
STYPE = 01 : Output Data FIFO	STYPE = 01 : reserved										
STYPE = 10 : Padding Block	STYPE = 10 : reserved										
STYPE = 11 : Out snooping*	STYPE = 11 : reserved										
13-12	Reserved.										
11-0 DL	Data Length. The number of bytes that will be passed to a CHA. A maximum of 12 bits is supported. This means for larger chunks of data multiple entries in the iNformation FIFO will be required.										
Format of bits 15-0 when STYPE is 10 (padding entries)											
15 PR	Prediction Resistance - If PTYPE specifies random data, setting PR=1 causes the RNG to supply random data with prediction resistance (i.e. reseeds the PRNG from the TRNG).										
14-12	Reserved. Must be 0.										
11 BM	Boundary Minus 1. When this bit is set with boundary padding, then boundary padding to a 4, 8 or 16-byte boundary minus 1 byte will be executed. For example, if a 16-byte boundary is selected with BM=1, padding will be done such that only 15 of the 16 bytes are used, leaving the 16th byte available for the user to fill.										
10 PS	Pad Snoop. When this bit is set then the Class 2 CHA will snoop the padding data from the Output Data FIFO rather than getting it from the padding block. When snooping, the Class 1 Alignment Block receives data from the Input FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.										
9-7	Reserved. Must be 0.										
6-0 PL	Pad Length. The number of bytes needed to pad the current data. If boundary padding is selected then this should be set to 4, 8 or 16 bytes. PL includes the length byte or zero byte for all zero last N, random last N and random last 0 padding types.										

### B.2.8.19 Input Data FIFO (CxIFIFO)

Data to be processed by the various CHAs is first pushed into the Input Data FIFO. Note that although the Input Data FIFO is 64-bits wide, a single 32-bit wide location is used to write data to the IFIFO. All data written to this location via the IP bus should be in big

endian format. The Input Data FIFO supports byte enables, allowing one to four bytes to be written to the IFIFO from the IP bus, or one to eight bytes via a descriptor. Although data is normally pushed in multiples of 8 bits, there is a special mechanism that allows a 4-bit value to be pushed into the Input Data FIFO. The IFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation CAAM will never overflow the Input Data FIFO. Care must be used to not overflow the Input Data FIFO when writing to it directly as results will be unpredictable. FIFO LOAD, FIFO STORE, LOAD, KEY, and MOVE commands can all automatically write to this register.

### **Figure B-147. Input Data FIFO - Format**

#### B.2.8.20 Output Data FIFO (CxOFIFO)

Data that is output from the various CHAs is pushed into the Output Data FIFO. The OFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation, CAAM will never overflow the Output Data FIFO. KEY, MOVE, MATH, and FIFO STORE commands will all read from the Output Data FIFO. Normally data is pushed in multiples of 8 bits, but there is a special mechanism that allows a 4-bit value to be pushed into the Output Data FIFO.

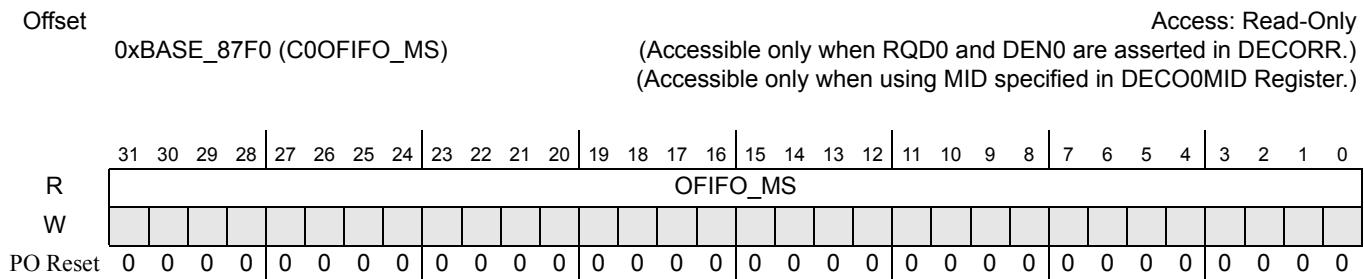
There are several commands that can result in data being pushed into the Output Data FIFO:

- The OPERATION Command can cause a Class 1 CHA to put data into the Output Data FIFO.
  - The KEY Command uses the Output Data FIFO when it decrypts keys. Since the Output Data FIFO must be empty and all transactions must have completed before the KEY Command will start, there will be no collision between a CHA push and an ODFNSR push to the Output Data FIFO.
  - The (SEQ) FIFO STORE Command, when encrypting keys, also pushes data into the Output Data FIFO.

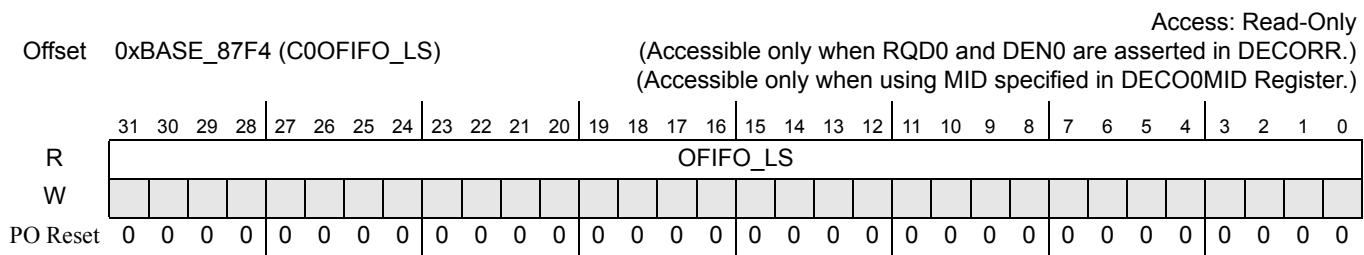
- A LOAD IMMEDIATE can push data directly into the Output Data FIFO. DECO will automatically stall a LOAD IMMEDIATE if necessary to prevent a collision with a push from the ODFNSR.
- A LOAD IMMEDIATE to the CHA Control Register can be used to “unload” the ARC4 S-Box into the Output Data FIFO.
- The (SEQ) FIFO STORE Command, when generating random data, also pushes data into the Output Data FIFO.

Automatic DECO stalling is accomplished as follows. Once the ODFNSR is written, the stalls described above will continue until the ODFNSR is cleared. That means that the Class 1 CHA has to assert its done signal and the ODFNSR has to have pushed its final value into the Output Data FIFO. **WARNING:** If the DECO is stalling while waiting for the ODFNSR to empty, and there is no already executed command (such as a FIFO STORE or MOVE) that will drain the Output Data FIFO sufficiently to allow the ODFNSR to empty, the DECO will hang.

Internally the Output Data FIFO is 64-bits wide, but since the IP bus is 32-bits wide, the Output Data FIFO is read via the IP bus using 32-bit word reads. The most-significant half of the 64-bit word is read from address 0xBASE\_x7F0, and the least significant half is read from address 0xBASE\_x7F4. All data read from the OFIFO is big endian.



**Figure B-148. Output Data FIFO\_MS - Format**



**Figure B-149. Output Data FIFO\_LS - Format**

### B.2.8.21 DECO Job Queue Control Register (DxJQCR)

This register tells the Descriptor Controller about the next Descriptor. During normal operation, this register is written by the Job Queue Controller. When a DECO is under the direct control of software, this register can be read or written from the IP Slave bus. Writing to the most-significant half of this register causes the Descriptor Controller to start processing. Note that at least the first burst of the Descriptor (including the Job Descriptor Header) must be written to the Descriptor buffer before the Job Queue Control Register is written.

Offset	Access: Read/Write															
	0xBASE_8800 (D0JQCR_MS)															
	(Accessible only when RQD0 and DEN0 are asserted in DECORR.)															
	(Accessible only when using MID specified in DECO0MID Register.)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	STEP	SING	WHL	FOUR	ILE	SHR_FROM				0	0	0	0	0	0	SOB
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	AMTD	0	0	0	0	SRC	0	0	0	0	0	0	ID			
PO Reset	0	0	0	0	0				0	0	0	0	0	0	0	0

**Figure B-150. DECO Job Queue Control Register\_MS (DxJQCR\_MS) - Format**

Offset	Access: Read/Write															
	0xBASE_8804 (D0JQCR_LS)															
	(Accessible only when RQD0 and DEN0 are asserted in DECORR.)															
	(Accessible only when using MID specified in DECO0MID Register.)															
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	CMD															
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	CMD															
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

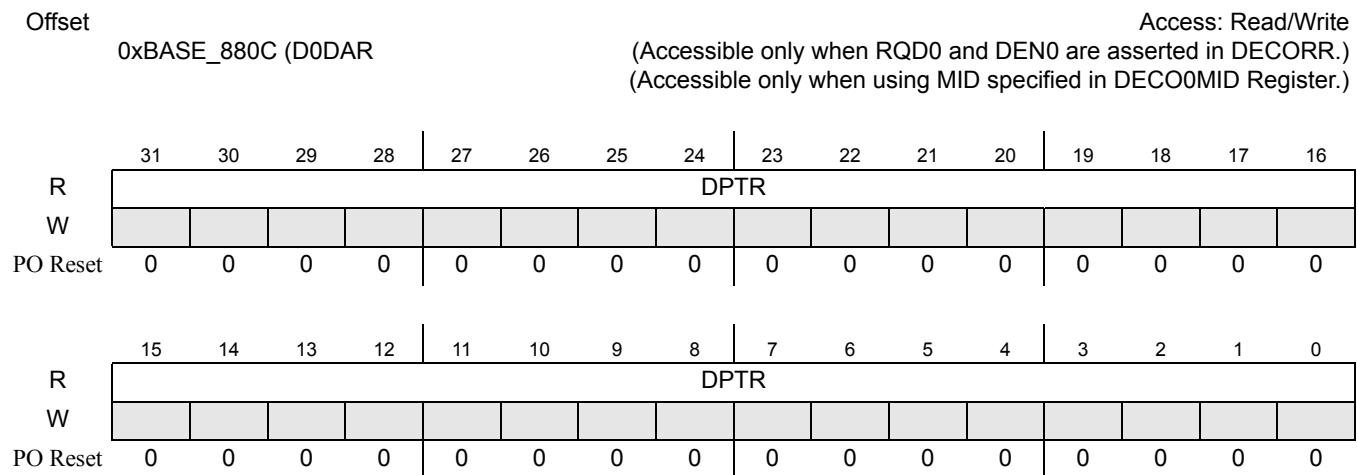
**Figure B-151. DECO Job Queue Control Register\_LS (DxJQCR\_LS) - Format**

**Table B-123. DECO Job Queue Control Register - Field Descriptions**

Field	Description
Most-significant portion of register	
31 STEP	Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO.
30 SING	Single Step Mode. This tells DECO to execute this descriptor, including jumps to nonlocal destinations, in single step mode. Only used by the processor that has control of DECO.
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the Job Queue Controller. This bit is set for certain Job Descriptors that are internally generated by CAAM.
28 FOUR	Four Words. Job Queue Controller is passing at least 4 words of the Descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. ILE = 0: No byte-swapping is performed for data transferred to or from the Descriptor Buffer. ILE = 1: Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.
26-24 SHR_FROM	Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the Job Queue Controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software).
23-17	Reserved.
16 SOB	Shared Descriptorburst. If set, the whole shared descriptor was passed to DECO with the Job Descriptor.
15	AMTD. Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRMID Register.
14	
13-11	Reserved.
10-8 SRC	Job Source. Source of the job. Determines which set of endian configuration bits that the DMA should use for transactions. It is illegal for the SRC field to have a value other than that of a Job Ring when running descriptors via the IP bus (i.e. under the direct control of software). 0b000: Job Ring 0 0b001: Job Ring 1 0b010: Reserved 0b011: Reserved 0b100: Reserved 0b101: Reserved 0b110: Reserved 0b111: Reserved
7-4	Reserved.
3-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.
Least-significant portion of register	
31-0 CMD	Command. In single-step mode, reading CMD returns the first word of the command that this DECO will execute next. This value is also readable via the STORE Command, but the value read is unpredictable.

### B.2.8.22 DECO Descriptor Address Register (DxDAR)

This DECO register holds the address of the currently executing Job Descriptor. When using DECO in single-step mode (see the CAAM Block Guide, Section “Executing Job Descriptors in Single-Step Mode”), this register must be written prior to the Job Queue Control Register.



**Figure B-152. DECO Descriptor Address Register\_LS (DxDAR) - Format**

**Table B-124. DECO Descriptor Address Register - Field Descriptions**

Field	Description
31-0 DPTR	Descriptor Pointer. Pointer to the Descriptor. Needed for write-back purposes.

### B.2.8.23 DECO Operation Status Register (DxOPSTA)

The DECO Operation Status Register is used to show DECO status following descriptor processing. This includes error conditions (if any), index of the next command within the descriptor, and condition flags set by Math Operations. Since the register is greater than 32

## CAAM Register Descriptions

bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

Offset	Access: Read/Write															
	(Accessible only when RQD0 and DEN0 are asserted in DECORR.)															
	(Accessible only when using MID specified in DECO0MID Register.)															
R	0xBASE_8810 (D0OPSTA_MS)															
W																
PO Reset																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	STATUS_TYPE				NLJ	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	COMMAND_INDEX						STATUS							
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-153. DECO Operation Status Register\_MS (DxOPSTA\_MS)

Offset	Access: Read/Write															
	(Accessible only when RQD0 and DEN0 are asserted in DECORR.)															
	(Accessible only when using MID specified in DECO0MID Register.)															
R	0xBASE_8814 (D0OPSTA_LS)															
W																
PO Reset																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	OUT_CT											
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	OUT_CT															
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-154. DECO Operation Status Register\_LS (DxOPSTA\_LS)

**Table B-125. DECO Operation Status Register - Field Descriptions**

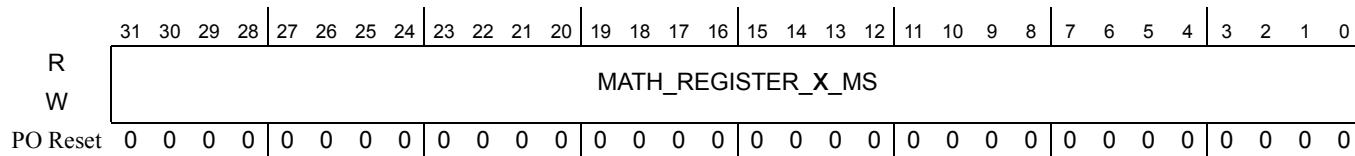
Field	Description																
most-significant portion of register																	
31-28 STATUS_TYPE	<p>Status Type. The type of status that is being reported for the just-completed command, defined as follows:</p> <table border="1" data-bbox="295 392 1470 772"> <tr> <td data-bbox="295 392 409 435">0000</td><td data-bbox="409 392 1470 435">no error</td></tr> <tr> <td data-bbox="295 435 409 477">0001</td><td data-bbox="409 435 1470 477">DMA error</td></tr> <tr> <td data-bbox="295 477 409 519">0010</td><td data-bbox="409 477 1470 519">CCB error</td></tr> <tr> <td data-bbox="295 519 409 561">0011</td><td data-bbox="409 519 1470 561">Jump Halt User Status</td></tr> <tr> <td data-bbox="295 561 409 604">0100</td><td data-bbox="409 561 1470 604">DECO error</td></tr> <tr> <td data-bbox="295 604 409 646">0101</td><td data-bbox="409 604 1470 646">Reserved</td></tr> <tr> <td data-bbox="295 646 409 688">0110</td><td data-bbox="409 646 1470 688">Reserved</td></tr> <tr> <td data-bbox="295 688 409 772">0111</td><td data-bbox="409 688 1470 772">Jump Halt Condition Code</td></tr> </table>	0000	no error	0001	DMA error	0010	CCB error	0011	Jump Halt User Status	0100	DECO error	0101	Reserved	0110	Reserved	0111	Jump Halt Condition Code
0000	no error																
0001	DMA error																
0010	CCB error																
0011	Jump Halt User Status																
0100	DECO error																
0101	Reserved																
0110	Reserved																
0111	Jump Halt Condition Code																
27 NLJ	Non-Local Jump. A jump was non-local. This includes non-local JUMP Commands and SEQ IN PTR RJD jumps and SEQ IN PTR INL jumps.																
26-14	Reserved. Always 0.																
13-8 COMMAND_INDEX	Command index: A pointer to a 32-bit word within the descriptor. If single stepping, this is the index of the next command to be executed. If not single stepping, this is the index of the command that is now executing. In the case of an error that is not a command problem, it is approximately the index of the command where the error occurred. If the error was due to a command problem, it is the index of the current command. A command problem is an error that is detectable by DECO as it executes the command (e.g. an illegal command type). Something that isn't a command problem is an error that occurs after the command has completed executing (e.g. illegal CHA modes, DMA errors, ICV check failures).																

Field	Description	
7-0 STATUS	If ERRTYP indicates no error, this field contains Math Status, as defined below:	
	7	Reserved
	6 GCD	Reserved
	5 PRM	Reserved
	4	Reserved
	3 MN	Math N. The result is negative. Can only be set by add and subtract functions, 0 otherwise
	2 MZ	Math Z. The result of a math operation is zero.
	1 MC	Math C. The math operation resulted in a carry or borrow.
	0 MNV	Math NV. Used for signed compares. This is a combination of the sign and overflow bits (i.e., Math N XOR Math C)
	If there was an error, this field contains Error Status, defined as in the Job Ring Output Status Register DESC ERROR field ( <a href="#">on page B-570</a> ).	
	least-significant portion of register	
31-29	Reserved	
28-0 OUT_CT	Output Count. Number of bytes written to sequential out pointer.	

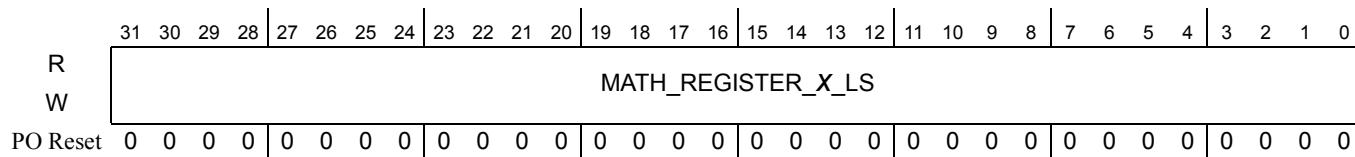
### B.2.8.24 Math Registers (DxMTH0, DxMTH1, DxMTH2 and DxMTH3)

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of four 64-bit registers. Data is moved into these registers via LOAD, MATH and MOVE commands. When a Math Register is accessed via the IP bus, the 64-bit Math Register is accessed as two 32-bit registers.

Offset	0xBASE_8840 (D0MTH0_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8848 (D0MTH1_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8850 (D0MTH2_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8858 (D0MTH3_MS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)

**Figure B-155. Math Register x\_MS (DxMTHx\_MS) - Format**

Offset	0xBASE_8844 (D0MTH0_LS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_884C (D0MTH1_LS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8854 (D0MTH2_LS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_885C (D0MTH3_LS)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)

**Figure B-156. Math Register x\_LS (DxMTHx\_LS) - Format**

### B.2.8.25 Gather Table Register (DxGTR) and Scatter Table Register (DxSTR)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. CAAM will fetch a burst's worth of entries at a time. A burst is 32 or 64 bytes, as indicated by the BURST field in the Master Configuration Register. Each register is 128 bits in length, so the data in each register is accessible over the 32-bit register bus as four 32-bit words.

Offset	0xBASE_8880 (D0GTR-0)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8900 (D0STR-0)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	Reserved (must be 0)
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 PO Reset 0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	Reserved (must be 0)

Figure B-157. Gather Table Register Word 0 and Scatter Table Register Word 0 - Format

Word 1 is located at the addresses shown below.

Offset	0xBASE_8884 (D0GTR-1)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8904 (D0STR-1)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 Address Pointer	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 Address Pointer	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-158. Gather Table Register Word 1 and Scatter Table Register Word 1 - Format**

Word 2 is located at the addresses shown below.

Offset	0xBASE_8888 (D0GTR-2)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_8908 (D0STR-2)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R W	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16 Length	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	
R W	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 Length	
PO Reset	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	

**Figure B-159. Gather Table Register Word 2 and Scatter Table Register Word 2 - Format**

Word 3 is located at the addresses shown below.

Offset	0xBASE_88C8 (D0GTR-3)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
Offset	0xBASE_890C (D0STR-3)	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)
R	31 30 29 28   27 26 25 24   23 22 21 20   19 18 17 16	Reserved (must be 0)
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	PO Reset
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0	Offset
W	0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0	PO Reset

**Figure B-160. Gather Table Register Word 3 and Scatter Table Register Word 3 - Format**

**Table B-126. Gather Table Register and Scatter Table Register - Field Descriptions**

Field	Description
Gather Table Register Word 0 and Scatter Table Register Word 0	
31-0	Reserved
Gather Table Register Word 1 and Scatter Table Register Word 1	
31-0 Address Pointer	This field holds the memory address to which this table entry points. This will either be a memory buffer (if E=0) or a Scatter/Gather Table entry (if E=1).
Gather Table Register Word 2 and Scatter Table Register Word 2	
31 E	Extension bit. If set to a 1, then Address Pointer refers to a Scatter/Gather Table entry instead of a memory buffer. It is an error to set the E bit if the SGT entry is a NULL buffer (i.e. Length and Address Pointer all 0s).
30 F	Final Bit. If set, this is the last entry of this Scatter/Gather Table.
29-0 Length	This field specifies how many bytes of data (for Gather Tables) or available space (for Scatter Tables) are located at the address pointed to by the Address Pointer.
Gather Table Register Word 3 and Scatter Table Register Word 3	
31-13	Reserved
12-0 Offset	Offset (measured in bytes) into memory where significant data is to be found. The use of an offset permits reuse of a memory buffer without recalculating the address.

### B.2.8.26 Descriptor Buffer (DxDESB)

The Descriptor Buffer is used by the DECO to buffer a Descriptor that has been fetched from memory. The Descriptor Buffer consists of 64 32-bit registers in consecutive addresses, beginning at the address shown below. For performance reasons, DECO doesn't execute the commands directly from the Descriptor Buffer. Instead, DECO executes commands from a four-word pipeline that is Dword-aligned with the Descriptor Buffer. Since commands vary in length from one to four words, up to three words in addition to the current command may also be resident in the pipeline. As a result, operations that modify the Descriptor Buffer may not have an immediate effect on the next few commands that execute. If the command being executed is in the first word of the pipeline (i.e. dword-aligned), then three more words are in the pipeline. If the command being executed is in the second word (i.e. word-aligned but not dword-aligned), then two more words are in the pipeline. Since commands vary from one to four words in length, these two or three words in the pipeline may contain additional commands that may be executed even if the corresponding words in the Descriptor Buffer are overwritten. (They won't be executed if the job terminates or the pipeline is flushed as described below.) To avoid anomalous behavior when overwriting the portion of the Descriptor Buffer containing the start of the currently executing command or the following two or three words, any commands in the pipeline that the programmer intends to execute should be completely contained within the pipeline.

There are several ways to flush the pipeline to ensure that recently loaded commands are executed rather than the pipeline-resident commands:

- Execute a JUMP command with a negative offset
- Use the JOB HEADER or SHARED HEADER commands to do an absolute jump.
- JUMP forward more than 3 words.

Offset	0xBASE_8A00 (D0DESB_<i>), offset 4, range 0 .. 63	Access: Read/Write (Accessible only when RQD0 and DEN0 are asserted in DECORR.) (Accessible only when using MID specified in DECO0MID Register.)																																																																		
R W	Descriptor_Buffer	<table border="1"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																			

Figure B-161. Descriptor Buffer - Format

### B.2.8.27 DECOx DBG\_JOB\_REG (DxDJR)

The DECOx DBG\_JOB\_REG, DECOx DBG\_DBG\_REG, DECOx DBG\_JOB\_PTR, DECOx DBG\_SHR\_PTR and DECOx DBG\_MID registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified. Note that the DECOx DBG\_JOB\_REG has the same format as the most-significant half of the DECO Job Queue Control Register.

Offset 0xBASE\_8E00 (D0DJR) - for DECO 0

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STEP	SING	WHL	FOUR	ILE	SHR_FROM				0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AMTD	JDMS	0	0	0	SRC				0	0	0	0	ID		
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-162. DECOx DBG\_JOB\_REG (DxDJR) - Format

Table B-127. DECOx DBG\_JOB\_REG - Field Descriptions

Field	Description
31 STEP	Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO.
30 SING	Single Step Mode. This tells DECO to execute this descriptor, including jumps to nonlocal destinations, in single step mode. Only used by the processor that has control of DECO.
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the Job Queue Controller. This bit is set for certain Job Descriptors that are internally generated by CAAM.
28 FOUR	Four Words. Job Queue Controller is passing at least 4 words of the Descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. ILE = 0: No byte-swapping is performed for data transferred to or from the Descriptor Buffer. ILE = 1: Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.

Field	Description
26-24 SHR_FROM	Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the Job Queue Controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software).
23-16	Reserved.
15 AMTD	Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRMID Register.
14 JDMS	Job Descriptor MID Select. Determines whether the SEQ MID or the Non-SEQ MID is asserted when reading the Job Descriptor from memory. 1 = SEQ MID 0 = Non-SEQ MID
13-11	Reserved.
10-8 SRC	Job Source. Source of the job. Determines which set of endian configuration bits that the DMA should use for transactions. It is illegal for the SRC field to have a value other than that of a Job Ring when running descriptors via the IP bus (i.e. under the direct control of software). 0b000: Job Ring 0 0b001: Job Ring 1 0b010: Reserved 0b011: Reserved 0b100: Reserved 0b101: Reserved 0b110: Reserved 0b111: Reserved
7-4	Reserved.
3-0 ID	Job ID. Unique tag given to each job by its source (see SRC field). Used to tell the source that the job has completed.

### B.2.8.28 DECOx DBG\_DBG\_REG (DxDDR)

The DECOx DBG\_JOB\_REG, DECOx DBG\_DBG\_REG, DECOx DBG\_PTR, DECOx DBG\_SHR\_PTR and DECOx DBG\_MID registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified.

Offset 0xBASE\_8E04 (D0DDR) - for DECO 0

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VALID	SD	TRCT		SEQMSEL	NSEQMSEL	DECO_STATE				PDB_WB_ST	PDB_STALL				
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PTCL_RUN	NLJ	CURRENT_REAL_ADDR				CMD_STAGE		CSA	NC	BWB	BRB	CT			
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-163. DECOx DBG\_DBG\_REG (DxDDR) - Format

Table B-128. DECOx DBG\_DBG\_REG - Field Descriptions

Field	Description
31 VALID	Valid.
30 SD	Shared Descriptor.
29-28 TRCT	Trans Count.
27-26 SEQMSEL	SEQ MID Select.
25-24 NSEQMSEL	Non-SEQ MID Select..
23-20 DECO_STAT E	DECO State.
19-18 PDB_WB_ST	PWB WB State.
17-16 PDB_STALL	PDB Stall.
15 PTCL_RUN	Protocol running delay.
14 NLJ	Took Non-local JUMP.
13-8 CURRENT_ REAL_ADDR	Current real address.
7-5 CMD_STAGE	Command Stage.

Field	Description
4 CSA	Command Stage Aux.
3 NC	No Command.
2 BWB	Burster Write Busy.
1 BRB	Burster Read Busy.
0 CT	Checking Trusted.

### B.2.8.29 DECOx DBG\_JOB\_PTR (DxDJP)

The DECOx DBG\_JOB\_REG, DECOx DBG\_DBG\_REG, DECOx DBG\_JOB\_PTR, DECOx DBG\_SHR\_PTR and DECOx DBG\_MID registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified. Note that DxDJP is a 64-bit register. When read from the IP bus it is read as two 32-bit words. When 32-bit pointers are in use the most-significant half of the register can be ignored.

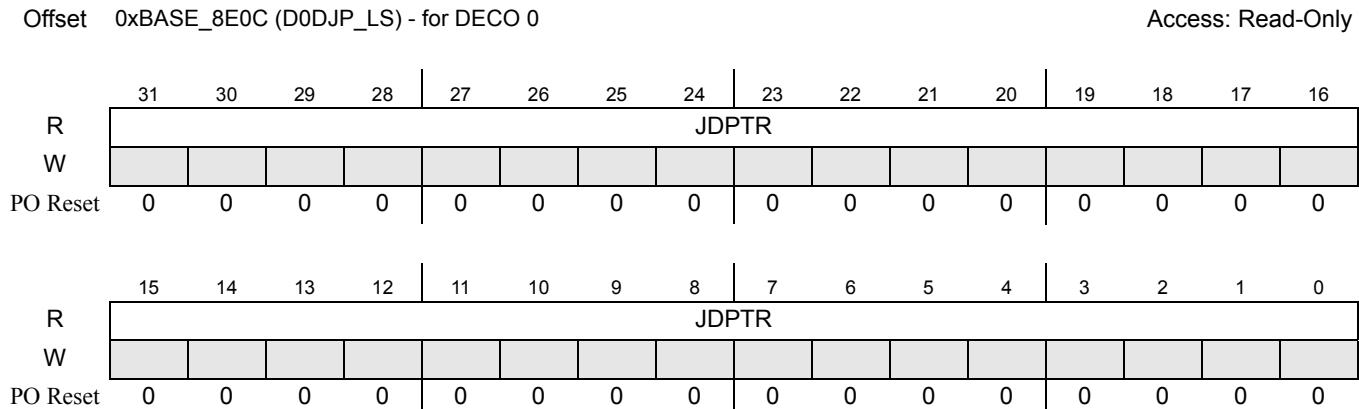
Offset 0xBASE\_8E08 (D0DJP\_MS) - for DECO 0 Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure B-164. DECOx DBG\_JOB\_PTR\_MS (DxDJP\_MS) - Format

**Figure B-165. DECOx DBG\_JOB\_PTR\_Ls (DxDJP\_LS) - Format****Table B-129. DECOx DBG\_JOB\_PTR - Field Descriptions**

Field	Description
31-0	Reserved.
31-0 JDPTR	Job Descriptor Pointer.

### B.2.8.30 DECOx DBG\_SHR\_PTR (DxSDP)

The DECOx DBG\_JOB\_REG, DECOx DBG\_DBG\_REG, DECOx DBG\_JOB\_PTR, DECOx DBG\_SHR\_PTR and DECOx DBG\_MID registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified. Note that DxSDP is a 64-bit register. When read from the IP bus it is read as two 32-bit words. When 32-bit pointers are in use the most-significant half of the register can be ignored.

Offset 0xBASE\_8E10 (D0SDP\_MS) - for DECO 0

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-166. DECOx DBG\_SHR\_PTR\_MS (DxSDP\_MS) - Format**

Offset 0xBASE\_8E14 (D0SDP\_LS) - for DECO 0

Access: Read-Only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SDPTR															
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SDPTR															
W																
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure B-167. DECOx DBG\_SHR\_PTR\_LS (DxSDP\_LS) - Format**

Field	Description														
<b>Most-significant portion of register</b>															
31-0	Reserved														
<b>Least-significant portion of register</b>															
31-0 SDPTR	Shared Descriptor Pointer														

### B.2.8.31 DECOx DBG\_MID\_REG (DxDMR)

The DECOx DBG\_JOB\_REG, DECOx DBG\_DBG\_REG, DECOx DBG\_JOB\_PTR, DECOx DBG\_SHR\_PTR and DECOx DBG\_MID registers are intended to assist in

## CAAM Register Descriptions

debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified. Note that the format of DxDMR is the same as the format of the DECO MID Status Register. This is a 64-bit register, so when read from the IP bus it is read as two 32-bit words.

Offset 0xBASE_8E18 (D0DLR_MS) - for DECO 0																Access: Read-Only			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure B-168. DECOx DBG\_MID\_REG\_MS - Format

Offset 0xBASE_8E1C (D0DLR_LS) - for DECO 0																Access: Read-Only			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
PO Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure B-169. DECOx DBG\_MID\_REG\_LS - Format

Field	Description
<b>Most-significant portion of register</b>	
31-20	Reserved.
19-16 DNSMID	DECO Non-SEQ MID. This is the MID that will be asserted during a DMA transaction that is related to descriptor operations other than Input or Output SEQ sequences.
15-6	Reserved.
5-0 DSMID	DECO SEQ MID. This is the MID that will be asserted during a DMA transaction that is related to an Input or Output SEQ sequence.

Field	Description
<b>Least-significant portion of register</b>	
31-4	Reserved.
3-0 DTMID	DECO Trusted MID. Any type of descriptor can write to this field, but only Trusted Descriptors can choose to assert this MID value during DMA transactions.

### B.2.8.32 Sequence Out Length (SOL) Register

The Sequence Out Length Register is used to specify the amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). The SOL Register is not accessible via the IP Bus, but can be read or written via the MATH Command.

not accessible via register bus (SOL0) - for DECO 0

Access: Read/Write (via the MATH Command)

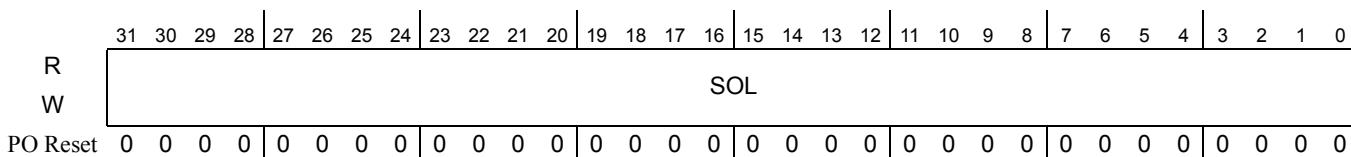


Figure B-170. Sequence Out Length (SOL) Register - Format

### B.2.8.33 Variable Sequence Out Length (VSOL) Register

The Variable Sequence Out Length Register is used to specify a variable amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). The VSOL Register is not accessible via the IP Bus, but can be read or written via the MATH Command.

not accessible via register bus (VSOL0) - for DECO 0

Access: Read/Write (via the MATH Command)

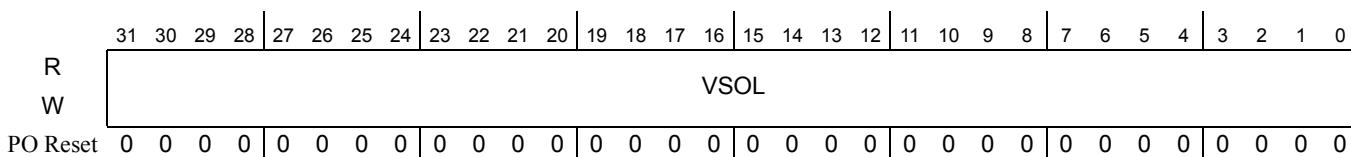
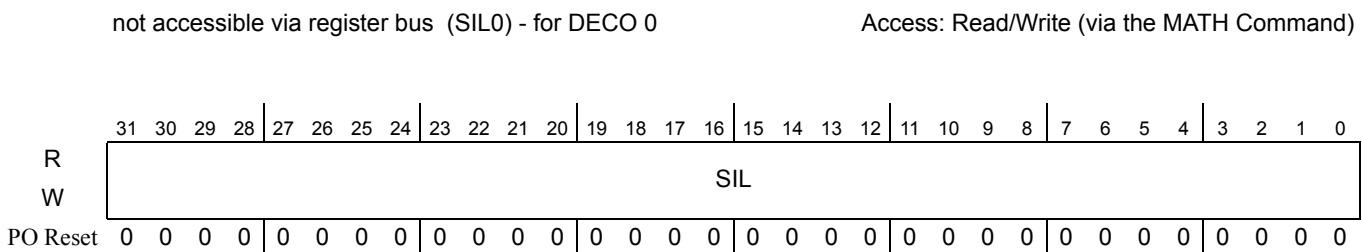


Figure B-171. Variable Sequence Out Length (VSOL) Register - Format

### B.2.8.34 Sequence In Length (SIL) Register

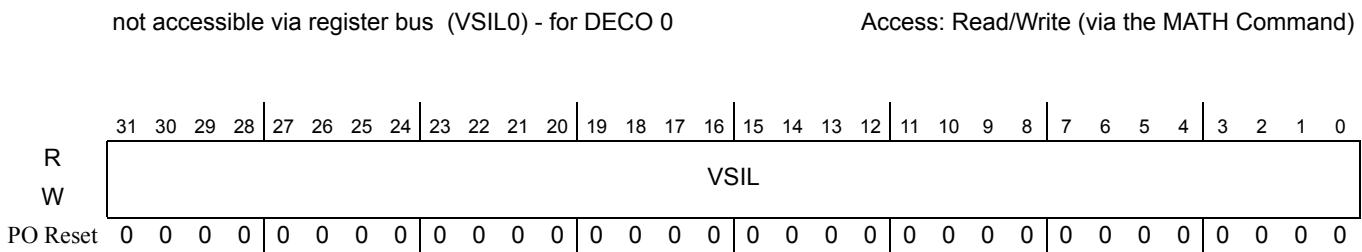
The Sequence In Length Register is used to specify the amount of data for an Input Sequence (a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). The SIL Register is not accessible via the IP Bus, but can be read or written via the MATH Command.



**Figure B-172. Sequence In Length (SIL) Register - Format**

### B.2.8.35 Variable Sequence In Length (VSIL) Register

The Variable Sequence In Length Register is used to specify a variable amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). The VSIL Register is not accessible via the IP Bus, but can be read or written via the MATH Command.



**Figure B-173. Variable Sequence In Length (VSIL) Register - Format**

**How to Reach Us:**

**Home Page:**

[www.freescale.com](http://www.freescale.com)

**Web Support:**

<http://www.freescale.com/support>

**USA/Europe or Locations Not Listed:**

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or  
+1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

**Europe, Middle East, and Africa:**

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
[www.freescale.com/support](http://www.freescale.com/support)

**Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku  
Tokyo 153-0064  
Japan  
0120 191014 or  
+81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

**Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 010 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale and the Freescale logo are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. ARM is the registered trademark of ARM Limited. ARM Cortex-A8 is the trademark of ARM Limited. Cadence Denali DDR portions Copyright © 2012, Cadence Design Systems, Inc., used with permission. Cadence is a registered trademark of Cadence Design Systems, Inc. All other product or service names are the property of their respective owners.  
© 2012 Freescale Semiconductor, Inc.



Document Number: IMX6DQ6SDLSRM  
Rev. D, 11/2012