Security Reference Manual for i.MX 8M Nano Applications Processor

Document Number: IMX8MNSRM
Rev. 0, 01/2020

Contents

Section number **Title** **Page**

Chapter 1
Disclaimer

Chapter 2
Security Overview

Chapter 3
Security System Integration

Chapter 4
System Boot

Chapter 5
Fusemap

Chapter 6
On-Chip OTP Controller (OCOTP_CTRL)

Chapter 7
Central Security Unit (CSU)

Chapter 8
Resource Domain Controller (RDC)

Chapter 9
TrustZone Address Space Controller (TZASC)

Chapter 10
Cryptographic Acceleration and Assurance Module (CAAM)

Chapter 11
Secure Non-Volatile Storage (SNVS)

Chapter 1
Disclaimer

1.1 Disclaimer

As system security requirements and the attack surface evolves, it is important for customers to understand the types of attacks (especially advanced physical attacks) which NXP does not claim to protect against, or strongly mitigate, so that appropriate mitigation can be taken by the customer at the system level if necessary:

- NXP does not guarantee against advanced tamper attempts, including the operation of the device beyond the defined specification limits. NXP does not guarantee the protection against semi-invasive and invasive attacks.
- This i.MX SoC has several built-in features addressing side channel attacks (e.g. mechanisms to make Differential Power Analysis (DPA) more difficult), however, there is no claim to be completely resistant. The effectiveness of these features has not been independently evaluated, therefore NXP does not guarantee that the result will meet specific customer requirements.
- This i.MX SoC's security trust architecture relies on the on the strength of cryptographic algorithms and digital signatures and if these are subsequently determined to have inherent flaws, then the impact for each flaw must be evaluated and, in this case, NXP does not guarantee the underlying trust architecture claims.
- This i.MX SoC has some built-in access control mechanisms to support the logical separation of executed code. However, NXP does not guarantee that the device completely ensures logical separation by itself. Any vulnerabilities identified in Trusted Execution Environments or Hypervisor software may impact this separation and data integrity and may require additional mitigations.

NXP recommends customers to implement appropriate design and operating safeguards based on defined threat models, to minimize the security risks associated with their applications and products.

Chapter 2
Security Overview

2.1 Overview

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

- Resource Domain Controller (RDC)
  - Supports 4 domains and up to 8 regions
- TrustZone (TZ) architecture—includes security extensions in the Cortex-A53 processors, Generic Interrupt Controller (GIC), TrustZone Watchdog (TZ WDOG), On-Chip RAM (OCRAM), and Trust Zone Address Space Controller (TZASC)
  - Arm Cortex-A53 MPCore TrustZone support
- On-chip RAM (OCRAM) secure region protection using OCRAM controller
- High Assurance Boot (HAB) feature in the system boot
- Cryptographic Acceleration and Assurance Module (CAAM):
  - Support Widevine and PlayReady content protection
  - Public Key Cryptography (PKHA) with RSA and Elliptic Curve (ECC) algorithms
  - Real-time integrity checker (RTIC)
  - DRM support for RSA, AES, 3DES, DES
  - True random number generation (RNG)
  - Manufacturing protection support
- Secure Non-Volatile Storage (SNVS)
  - Secure real-time clock (RTC), key storage, and security monitor
- Secure JTAG Controller (SJC) with secure debug
- On-chip One-Time Programmable Element Controller (OCOTP_CTRL) with on-chip electrical fuse arrays
- Central Security Unit (CSU)

2.2 Feature summary

This figure shows a simplified diagram of the security subsystem:

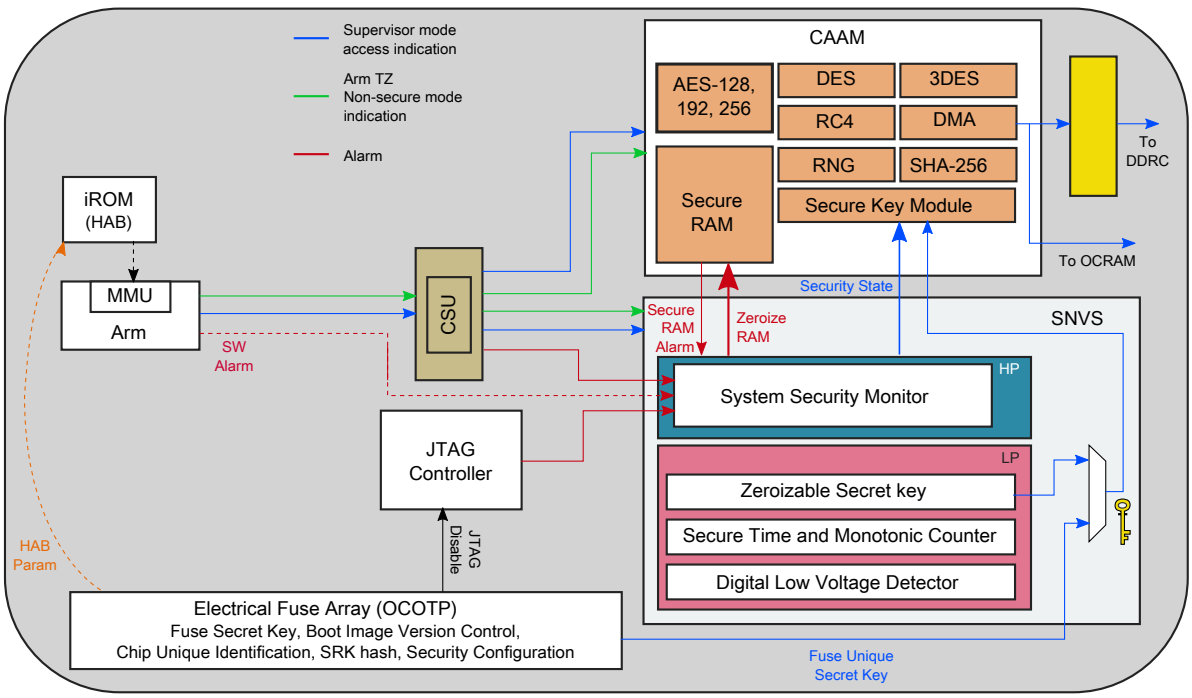


Figure 2-1. Security subsystem (simplified)

This diagram represents an example of the CSU use.

All platforms built using this chip share a general need for security, though the specific security requirements vary greatly from platform to platform. For example, portable consumer devices need to protect a different type and cost of assets than the automotive or industrial platforms. Each market must be protected against different kinds of attacks. The platform designers need an appropriate set of counter measures to meet the security needs of their specific platform.

To help the platform designers to meet the requirements of each market, the chip incorporates a range of security features. Most of these features provide protection against specific kinds of attack, and can be configured for different levels according to the required degree of protection. These features are designed to work together or independently. They can be also integrated with the 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 of access from the CPU and DMA peripherals to the on-chip peripherals and to both the on-chip and off-chip memories (see CSU, RDC, AIPSTZ)
    - Interrupt separation
    - Secure storage separation
    - Cryptographic separation
    - Encrypted Boot
  - Manufacturing protection
- High-Assurance Boot
  - Security library embedded in the 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 the OTP fingerprint (SHA-256)
  - Runs every time the chip is reset
  - Image version control/image revocation (on-chip OTP-based)
- Secure Non-Volatile Storage (SNVS)
  - On-chip zeroizable secure RAM (32 KB)
  - Off-chip storage protection using AES-256 and the chip's die-individual hardware-only key
- Hardware cryptographic accelerators
  - Symmetric: AES-128/192/256, DES/3DES
  - Asymmetric: RSA (up to 4096), Elliptic Curve (up to 1023)
  - Hash message digest and HMAC: SHA-1, SHA-256, MD-5
  - Built-in Protocols: ECDSA/DSA
- True and pseudorandom number generator
- On-chip secure real-time clock with autonomous power domain
- Secure debugging
  - Configurable protection against unauthorized JTAG manipulation
  - Three security levels + a complete JTAG disable
  - Support for JTAG port secure reopening for field return debugging
- Electrical fuses (OTP Memory)

2.3 TrustZone architecture

The TrustZone architecture provides a trusted execution environment for security-critical software. The 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 the security services and sensitive data. Furthermore, the security services cannot be starved of the access to the processor resources or hijacked by uncontrolled interrupts. TrustZone enables the security-critical software to coexist with a rich platform software environment.

These features work together to create the TrustZone architecture:

- The TrustZone security extensions in the Arm core (see Figure 2-2) duplicate the user, supervisor, and other privileged modes of the processor in the Secure World and the Normal World. The security services execute in the Secure World under the control of a security kernel, while the normal services and applications run in the Normal World with a rich operating system. TrustZone also provides a monitor mode, to which the Normal World operating system traps when the security services are required.
- The TrustZone extensions to the MMU and memory caches separate the 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. Using this feature, the security kernel controls the access to the Secure World memory and peripherals from the Normal World software (even the operating system).
- TrustZone Access Controller (TZASC) separates the Secure-World and Normal-World external memory spaces for other bus masters such as the DMA-equipped peripherals. For more details on TZASC, see TrustZone Address Space Controller (TZASC).
- The On-Chip RAM (OCRAM) controller separates the Secure-World and Normal-World internal memory spaces for other bus masters. For more details on OCRAM, see the OCRAM section of the reference manual.
- Central Security Unit (CSU) and the peripheral bridge (AIPS-TZ) separates the Secure-World and Normal-World peripheral address spaces for other bus masters, such as the DMA-equipped peripherals. For more details on CSU, see Central Security Unit (CSU). For more information on AIPSTZ (in addition to what is presented in this document), see the chip reference manual.
- Generic Interrupt Controller (GIC) collects the 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 the Secure World software. The context switch to handle the normal or secure interrupts when executing in the Normal World or the Secure World is configured in the Arm core.

- TrustZone Watchdog (TZ WDOG) protects against the Normal World software preventing to switch back to the Secure World, thereby starving the security services of the access to the core. When TZ WDOG is activated, the Secure World software must service it on a periodic basis. If the servicing does not take place before the configured timeout, TZ WDOG asserts a secure interrupt that forces a switch to the Secure World. If it is still not served, TZ WDOG asserts a security violation alarm. TZ WDOG cannot be programmed or deactivated from the Normal World. For more details on TZ WDOG, see TrustZone Watchdog (TZ WDOG).

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

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

The following figure shows the four execution modes.

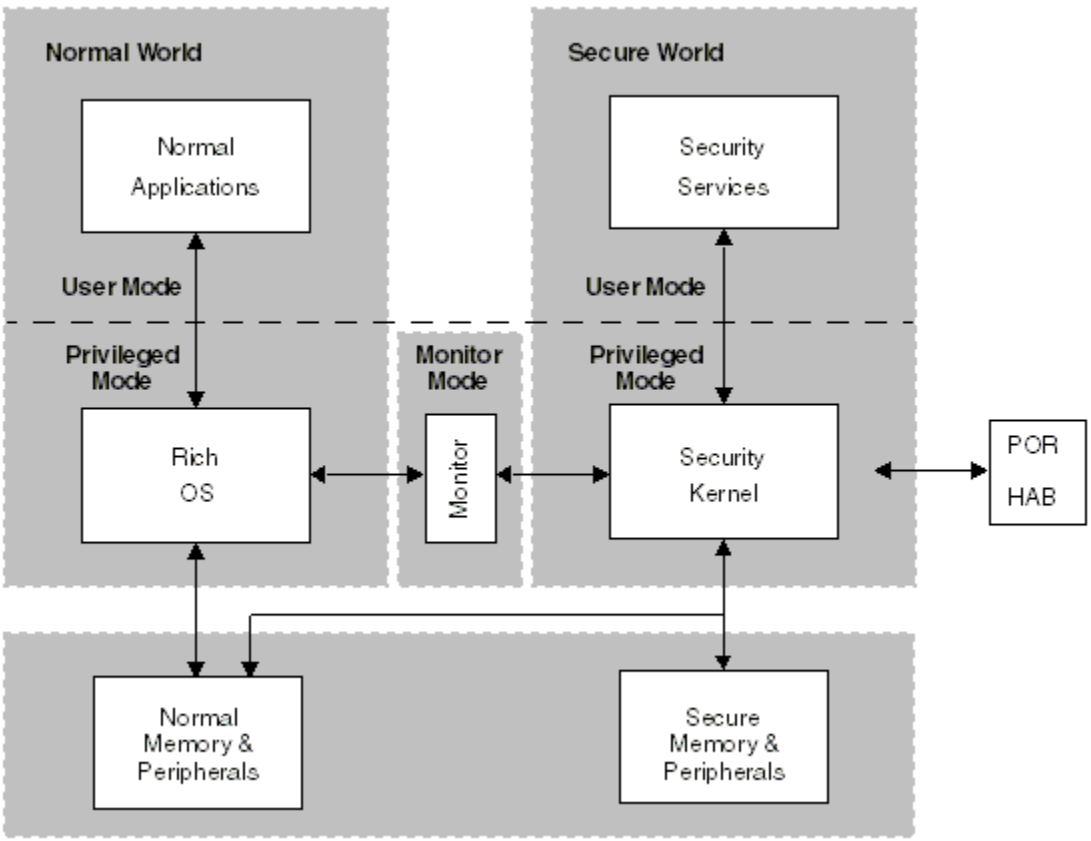


Figure 2-2. TrustZone security extensions

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 is secure and the other is non-secure.

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

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

- The CSU configures other bus masters to make either Secure World or Normal World accesses.
- Secure storage separation .
- 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 A53 Platform in the chip reference manual.

2.4 High-Assurance Boot (HAB)

The HAB, which is the high-assurance boot feature in the system boot ROM, detects and prevents the execution of unauthorized software (malware) during the boot sequence.

When the unauthorized software is permitted to gain control of the boot sequence, it can be used for a variety of goals, such as exposing stored secrets; circumventing access controls to sensitive data, services, or networks, or for repurposing the platform. The unauthorized software can enter the platform during upgrades or reprovisioning, or when booting from the USB connections or removable devices.

The HAB protects against unauthorized software by:

- Using digital signatures to recognize the authentic software. This enables you to boot the device to a known initial state and run the software signed by the device manufacturer.

2.4.1 HAB process flow

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

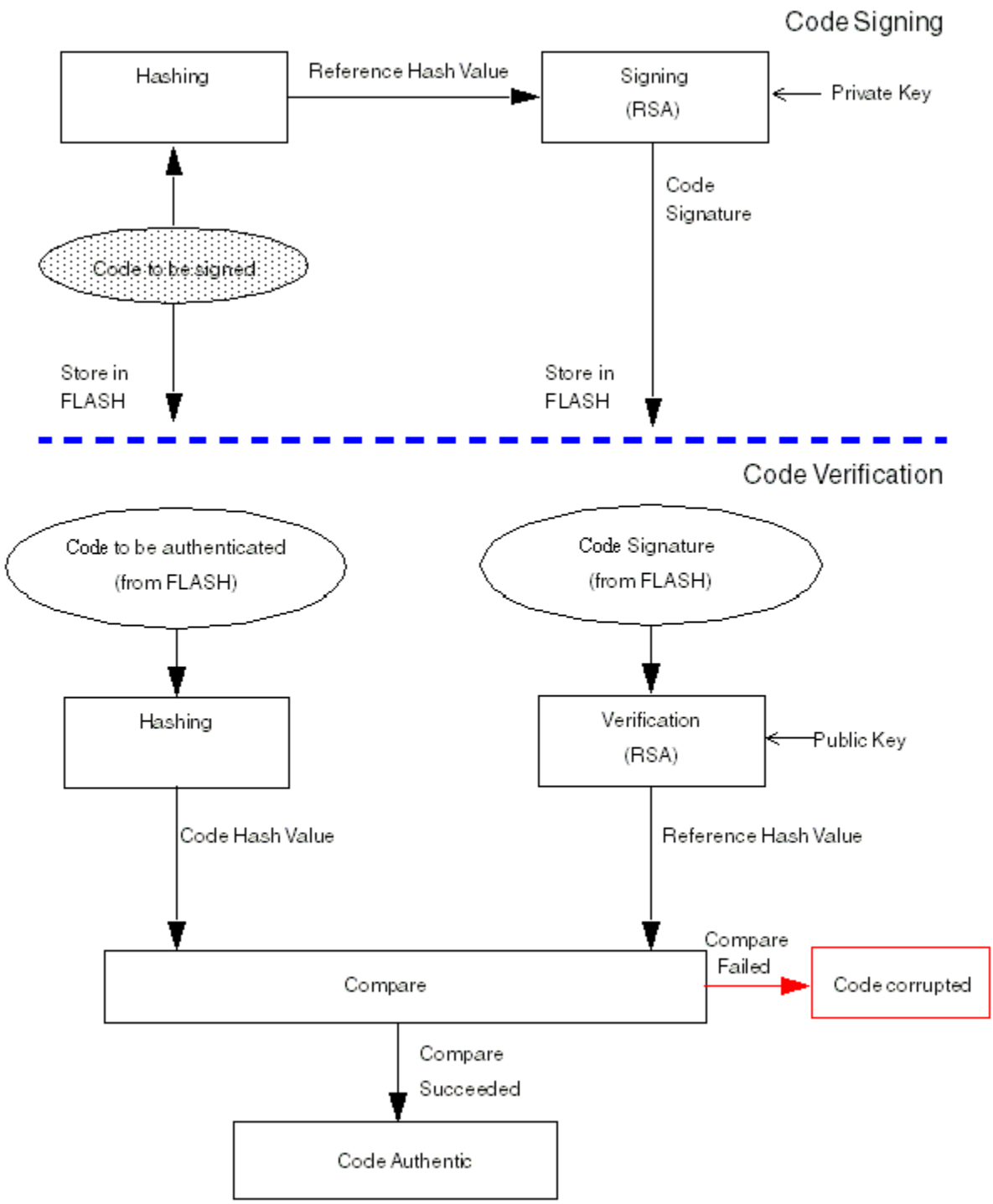


Figure 2-3. Code signing and authentication processes

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

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

2.4.2 HAB feature summary

The HAB features:

- Enforced internal boot via on-chip masked ROM
- Authentication of software loaded from any boot device (including the USB download)
- CMS PKCS#1 signature verification using the RSA public keys (from 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 the manufacturer-programmable on-chip fuses
- Multiple root public keys with revocation by fuses
- Initialization of other security components
- Authenticated USB download fall-over on any security failure
- Open configuration for development purposes and non-secure platforms
- Closed configuration for shipping secure platforms

On the chip, the HAB is integrated with these security features:

- The HAB executes in the TrustZone Secure World.
- The HAB initializes the SNVS security monitor state machine. A successful secure boot with the HAB is required for the platform software to gain access to use the master secret key selected by SNVS.
- The HAB reads the root public key fingerprint, revocation mask, and security configuration from the OCOTP_CTRL.
- The HAB can use the to accelerate hash calculations.

2.5 Secure Non-Volatile Storage (SNVS) module

- Provides a non-volatile real-time clock maintained by an uninterrupted power source during system power down for use in both the secure and non-secure platforms.
- 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 revocations

- Controls the access to the OTP master secret key used by the to protect confidential data in the off-chip storage
- Provides non-volatile highly protected storage for an alternative master secret key

2.5.1 SNVS architecture

The 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 isolation cells which are library-instantiated cells that ensure that the powered-up logic is not corrupted when the power goes down in the rest of the chip.

The SNVS_LP has these functional units:

- Zeroizable Master Key
- Secure non-rollover real-time counter with alarm
- Non-rollover monotonic counter
- Digital Low-voltage Event Detection
- General-purpose register
- Control and status registers

The SNVS_HP is in the chip power-supply domain. The SNVS_HP provides an interface between the SNVS_LP and the rest of the system. The access to the SNVS_LP registers can be gained through the SNVS_HP only when it is powered up according to the access permission policy.

The SNVS_HP has these 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

2.6 Cryptographic Acceleration and Assurance Module (CAAM)

The 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
- A secure, hardware random number generation provided by the RNG4-Pseudorandom Number Generator (PRNG), designed to be compliant with the National Institute of Standards and Technology (NIST)
- Export and import of cryptographic blobs
- Secure memory controller and interface
  - Protection for the cryptographic keys against the exposure to malicious software running on the processor
  - Protected storage for confidential data, such as the proprietary software, encryption keys, passwords, or PINs against unauthorized disclosure (including cloning)
  - Protected storage for trusted data, such as the Digital Rights Management (DRM) licences, biometrics reference data, or root certificates against unauthorized modification
  - On-chip (run-time) and off-chip protected storage
- Automatic zeroization of the security parameters and the secure memory
- IP slave interface
- DMA

2.7 OCOTP_CTRL

The OCOTP_CTRL provides the primary user-visible mechanism for interfacing with the on-chip fuses. These fuses' uses include:

- Unique chip identifiers
- Mask revision numbers
- Cryptographic keys
- Security configuration
- Boot characteristics
- Various control signals requiring permanent non-volatility

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

The OCOTP_CTRL provides:

- Shadow cache of fuse values, loaded at reset, before the system boot
- Ability to read the fuse values in the shadow cache (does not affect the fuse element)

- Ability to read the fuses directly (ignoring the shadow cache)
- Ability to write (program) the fuses by software or JTAG
- Fuses and shadow cache bits enforce 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).

2.8 Central Security Unit (CSU)

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

The CSU provides:

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

On the chip, the CSU interfaces with:

- The Arm TrustZone architecture to assign peripherals to the Secure World and Normal World domains

2.9 Resource Domain Controller (RDC)

The Resource Domain Controller (RDC) provides a mechanism to allow boot time configuration code to establish resource domains by assigning cores, bus masters, peripherals and memory regions to domain identifiers. Once configured, bus transactions are monitored to restrict accesses initiated by cores and bus masters to their respective peripherals and memory.

For shared peripherals, the RDC provides a semaphore-based locking mechanism to provide for temporary exclusivity while the domain software uses the peripheral. Once the software of one domain has finished the task and finished with the peripheral then it may release the semaphore making the peripheral available to the other domain.

The RDC provides:

- Assignment of cores, bus masters, peripherals, and memory regions to a resource domain
- Fixed memory resolution of 128 Bytes for small address spaces and 4 KB for large address spaces
- Four resource domain identifiers
- Memory read/write access controls for each resource domain and region
- Optional semaphore-based, hardware-enforced exclusive access of shared peripherals to a resource domain
- Prioritized access permissions for overlapping memory regions
- Automatic restoration of resource domain access permissions to memory regions in the power-down domain

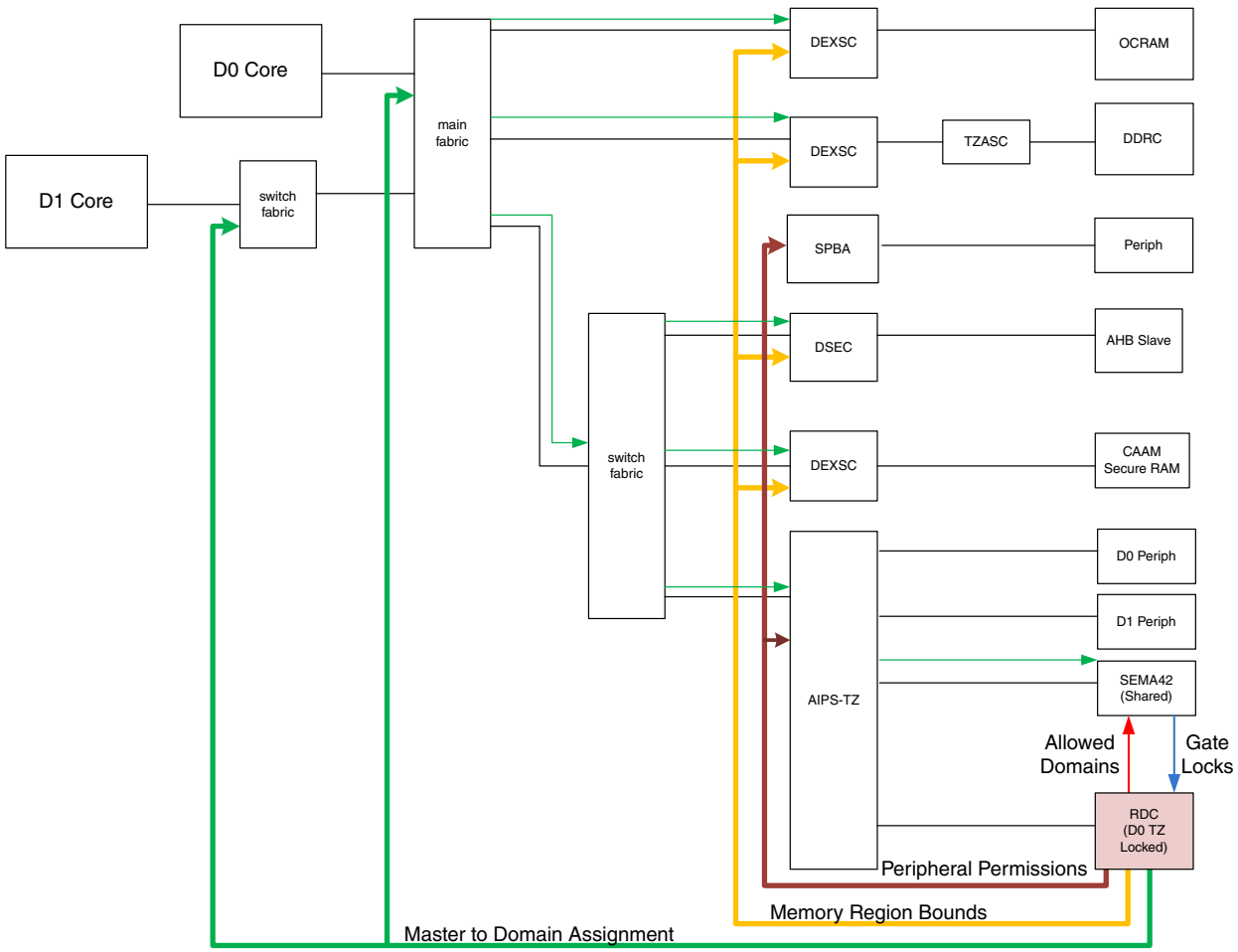


Figure 2-4. Example RDC Connections

2.10 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.

Masters and peripherals are assigned to one or more resource domains in the RDC submodule (see the RDC chapter for details). Depending on RDC programming, masters transactions through the AIPSTZ may or may not be allowed access to peripherals in different resource domains.

2.11 System JTAG Controller (SJC)

The JTAG port provides debug access to hardware blocks, including the Arm processor and the system bus. This enables 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. The System JTAG Controller (SJC) protects against unauthorized JTAG manipulation. It also provides a JTAG port that conforms to the IEEE 1149.1 and IEEE 1149.6 (AC) standards for BSR (boundary-scan) testing.

The SJC provides these security levels:

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

The security levels are selected via the e-fuse configuration.

NOTE

For the final production hardware, the most secure and preferred method is to ensure that the JTAG interface is not pinned or routed out at all. The same applies to any other trace, diagnostic or test ports including SDP.

2.11.1 Scan protection

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

- The chip is reset when entering the scan mode.
- All modules are reset two clock cycles before receiving the scan-enable indication.

- The chip cannot exit the scan mode without a reset.
- The security modules (including CAAM, SNVS, CSU, and OCOTP_CTRL) have an additional scan-protection logic to protect the sensitive internal data and functionality.

See the "System JTAG Controller (SJC)" section in for more information on the SJC.

2.12 TrustZone Address Space Controller (TZASC)

The 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 programmable independently for each address region and permits the data transfers between the 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.
- Allows the host interrupt to be programmed to signal the attempted access control violations.

For a detailed specification of TZASC, see the Arm® Infocenter documentation center, Revision: r0p1 *CoreLink™ TrustZone Address Space Controller TZC-380 Technical Reference Manual*.

2.13 Smart Direct Memory Access Controller (SDMA)

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

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

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

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

For more details on the SDMA, see the "System DMA (SDMA)" section in the chip reference manual.

2.14 TrustZone Watchdog (TZ WDOG)

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

When the TZ WDOG module is activated, it must be serviced by the Secure World software on a periodic basis. If the servicing does not take place, the timer expires and the TZ WDOG asserts a secure interrupt that forces the processor to switch to the 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 the Normal World software.

The TZ WDOG is another instantiation of the system WDOG. The TZ WDOG has these 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 the low-power modes
- Configurable counters to run or stop during the debug mode
- Two event time points: one for the TrustZone interrupt assertion and one for the security alarm assertion

Chapter 3
Security System Integration

3.1 Master ID allocation

The master IDs are used in setting up the TrustZone Address Space Controller (TZASC) and in profiling. This table summarizes the master IDs for all system master modules:

Table 3-1. Master IDs

Module	Master ID
Cortex A53	001b
SDMA	011b
All others	000b

3.2 System-level SNVS connections

3.2.1 System security violation alarm signals monitored by SNVS

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

Table 3-2. SNVS system security violation alarm input signals

SNVS registers			Source	Description
HPSVSR register bit field	HPSVCR register bit field	LPSVCR register bit field		
bit 0 - CAAM	bit 0 - CAAM_CFG	bit 0 - CAAM_EN	CAAM	CAAM security violation
bit 1 - SJC	bit 1 - SJC_CFG	bit 1 - SJC_EN	SJC	JTAG active
bit 2 - WDOG2	bit 2 - WDOG2_CFG	bit 2 - WDOG2_EN	WDOG2	Watchdog 2 reset

Table continues on the next page...

Table 3-2. SNVS system security violation alarm input signals (continued)

SNVS registers			Source	Description
HPSVSR register bit field	HPSVCR register bit field	LPSVCR register bit field		
bit 3 - Reserved	bit 3 - Reserved	bit 3 - Reserved	(Reserved)	—
bit 4 - SRC	bit 4 - SRC_CFG	bit 4 - SRC_EN	SRC	Internal boot
bit 5 - Reserved	bit 5 - Reserved	bit 5 - Reserved	(Reserved)	—

3.3 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. The SEC_RRR_RESP itself can be locked, preventing further modifications by the LOCK_SEC_ERR_RESP bit (GPR10[27]) to assure the system security integrity.

For more information, see the general registers in the chip reference manual.

3.4 OCRAM TrustZone support

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

When the 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) in accordance with 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 the secure or non-secure modes. The secure-region addresses are programmed through the IOMUXC and should be modified when no transactions are occurring on the OCRAM bus.

The TrustZone bits are described in the "Programmable Registers" section of the chip reference manual.

NOTE

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

This figure shows the OCRAM schematic connectivity:

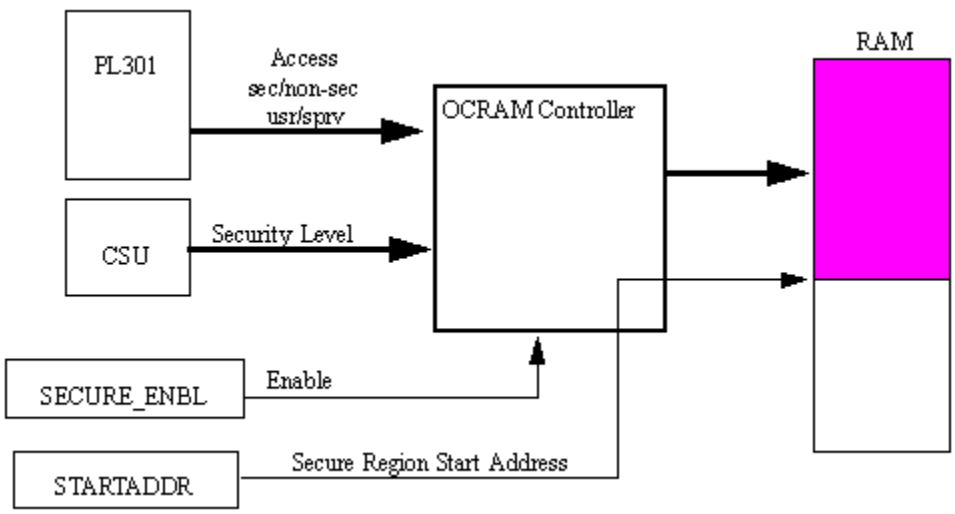


Figure 3-1. OCRAM schematic connectivity

3.5 Watchdog mechanism

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

The WDOG module operates as follows:

- If servicing does not take place, the timer times out and asserts the internal system reset signal (wdog_rst_B) which goes to the SRC (System Reset Controller).
- The interrupt can be generated before the timer actually times out.
- The 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.

This figure describes the WDOG1 and WDOG2 connectivity at the system level.

Figure 3-2. WDOG1 and WDOG2 alarms and interrupts connectivity

3.6 Security configuration

The following figure illustrates the typical device security-configuration lifecycle which starts from the IC fabrication and continues with the 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 or NXP facilities. Note that the field-return configuration is required for NXP to run test patterns even on non-secure products (open configuration).

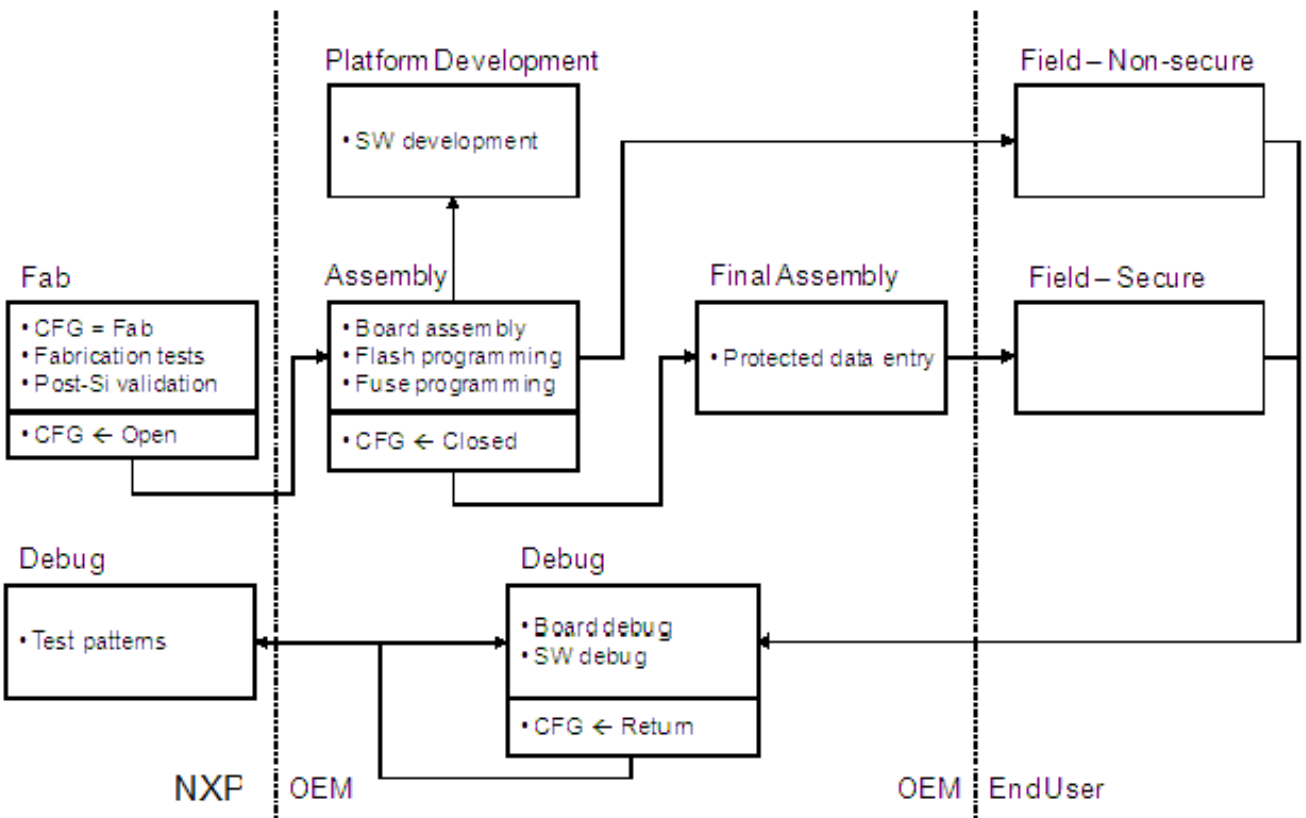


Figure 3-3. Device security configuration life cycle

3.6.1 Field return for retest procedure

Manufacturers can enable the debugging restrictions designed to protect the device keys and other sensitive data on the devices shipped to the end users. These debugging restrictions include these measures:

- Disabling JTAG (using JTAG_SMODE+KTE, or SJC_DISABLE fuses).
- Preventing the execution of unauthorized bootloader software (using the SEC_CONFIG[1] fuse).

Naturally, these debugging restrictions also constrain the legitimate debugging of the field-return devices with suspected faults. The chip includes a field-return configuration to enable the legitimate debugging, including the possibility for NXP to run the test modes on returned parts. The field-return configuration:

- Enables JTAG (overriding JTAG_SMODE+KTE and SJC_DISABLE fuses).
- Enables the execution of unsigned bootloader software as in the open configuration (overriding the SEC_CONFIG[1] fuse).

To protect the sensitive data already provisioned, the field-return configuration permanently disables the access to the device keys (including access from the or DTCP modules).

The entry to the field-return configuration is strictly controlled 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 (provided the OCOTP clock has been enabled in the initial bootloader either via the DCD or plug-in method), so that the FIELD_RETURN fuse cannot be burned.
- Setting the FIELD_RETURN_LOCK bit can be avoided by including the unlock command in the CSF or DCD (open configuration only) which provides:
  - The CSF and bootloader software pass signature verification (Closed configuration).
  - The unlock command arguments match the value in the UNIQUE_ID fuses on the device.

NOTE

OCOTP_CTRL fuse controller clocks should be enabled for the FIELD_RETURN functionality.

The typical mass-production bootloader on the shipped devices has no unlock command, so the entry to the field-return configuration requires booting to a special bootloader which is customized for a single device. For the closed devices, a special bootloader must be signed for that single device, so that it cannot be used to unlock other devices even if it leaks to the end users.

The boot flow to activate the field-return configuration is shown in this figure:

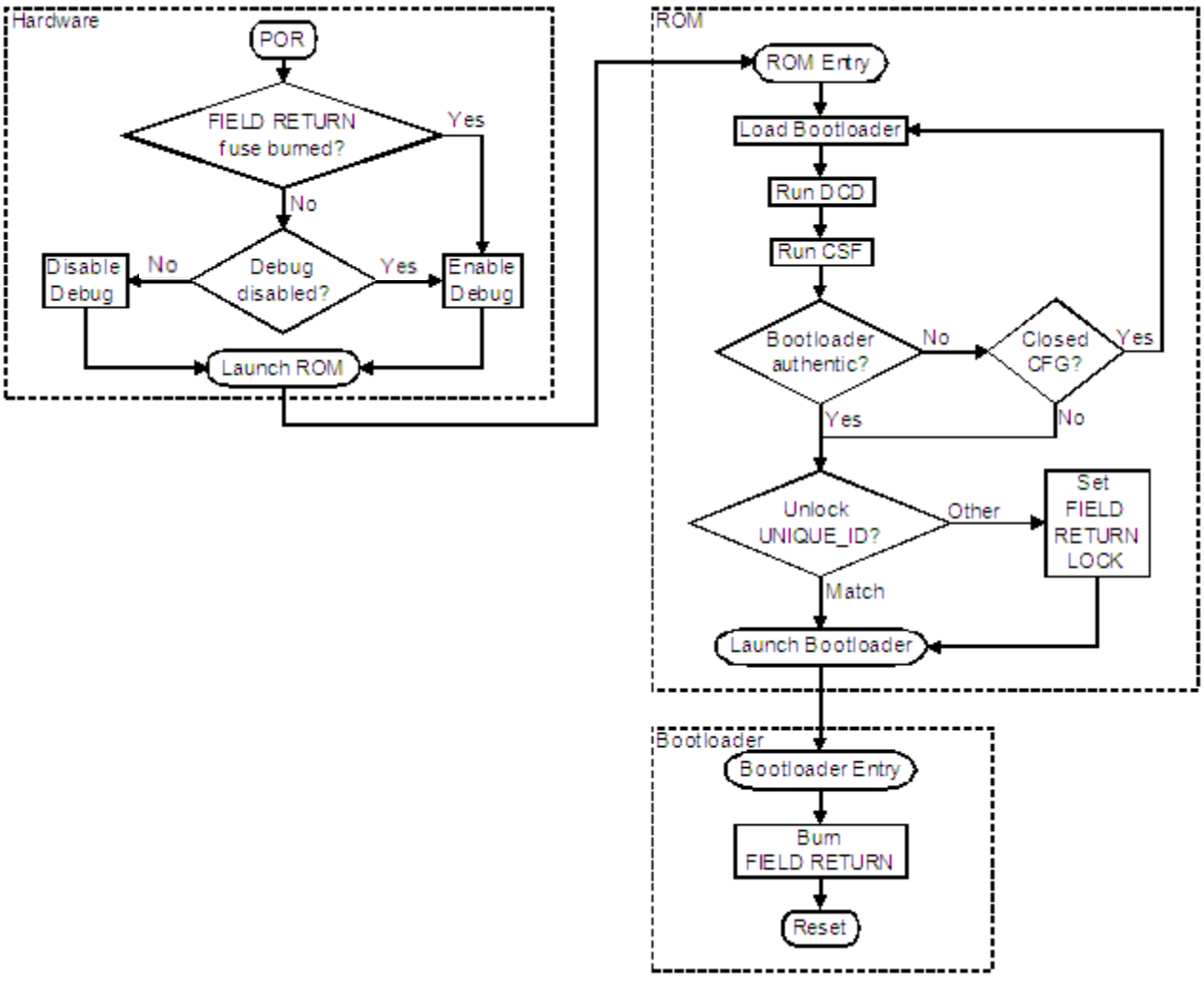


Figure 3-4. HAB FIELD_RETURN flowchart

When the FIELD_RETURN fuse is burned, the part is nearly returned to its open state after the next POR, including:

- Enabling the JTAG (overriding blocking by other means).
- Enabling the unsigned images to execute, as in the open configuration.
- Block access to the sensitive keys provided by the OCOTP_CTRL or SNVS to CAAM.
- Note that the field-return configuration is required for NXP to run the test patterns even on non-secure products (open configuration).

Chapter 4
System Boot

4.1 Overview

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

The boot ROM code uses the state of the internal register BOOT_MODE[3:0] 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)

The boot ROM supports these boot devices:

- Serial NOR Flash via FlexSPI
- NAND flash
- SD/MMC
- Serial (SPI) NOR

The boot ROM uses the state of the BOOT_MODE and eFUSEs to determine the boot device. For development purposes, the eFUSEs used to determine the boot device may be overridden using the GPIO pin inputs.

The boot ROM code also allows to download the programs to be run on the device. The example is a provisioning program that can make further use of the serial connection to provide a boot device with a new image. Typically, the provisioning program is downloaded to the internal RAM and allows to program the boot devices, such as the SD/MMC flash. The ROM serial downloader uses a high-speed USB in a non-stream mode connection.

A key feature of the boot ROM is the ability to perform a secure boot, also known as a High-Assurance Boot (HAB). This is supported by the HAB security library which is a subcomponent of the ROM code. The HAB uses a combination of hardware and software together with the Public Key Infrastructure (PKI) protocol to protect the system from executing unauthorized programs. Before the HAB allows the user 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 a 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. 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 the API functions, allowing the additional boot chain components (bootloaders) to extend the secure boot chain. The out-of-fab setting for the SEC_CONFIG is the open configuration, in which the ROM/HAB performs the image authentication, but all authentication errors are ignored and the image is still allowed to execute.

4.2 Boot modes

During boot, the core's behavior is defined by the boot mode pin settings, as described in Boot mode pin settings.

4.2.1 Boot mode pin settings

The device supports 16 boot modes but only several are supported on the chip and the others are reserved for future use. The boot mode is selected based on the binary value stored in the internal BOOT_MODE register.

The BOOT_MODE is initialized by sampling the BOOT_MODE inputs on the rising edge of the 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 this table for settings:

Table 4-1. Boot MODE pin settings

Boot Device Select	BOOT_MODE [5]	BOOT_MODE [4]	BOOT_MODE [3]	BOOT_MODE [2]	BOOT_MODE [1]	BOOT_MODE [0]
Boot from internal fuses	0	0	0	0	0	0

Table continues on the next page...

Table 4-1. Boot MODE pin settings (continued)

Boot Device Select	BOOT_MODE [5]	BOOT_MODE [4]	BOOT_MODE [3]	BOOT_MODE [2]	BOOT_MODE [1]	BOOT_MODE [0]
USB Serial Download	0	0	0	0	0	1
USDHC3 (eMMC)	0	0	0	0	1	0
USDHC2 (SD)	0	0	0	0	1	1
NAND 8-bit single device, 256 pages	0	0	0	1	0	0
NAND 8-bit single device, 512 pages	0	0	0	1	0	1
FlexSPI 3B Read	0	0	0	1	1	0
FlexSPI Hyperflash 3.3V	0	0	0	1	1	1
eCSPI Boot	0	0	1	0	0	0
Reserved	0	0	1	0	0	1

4.2.2 High-level boot sequence

The figure found here show the high-level boot ROM code flow.

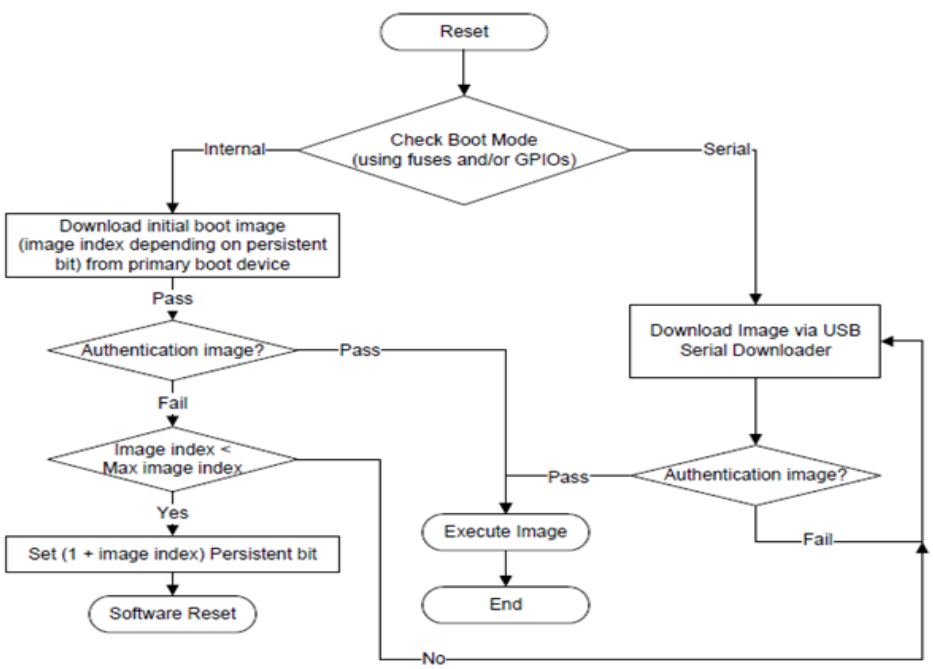


Figure 4-1. Boot flow

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) was not programmed yet, 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 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 the Boot From Fuses mode addresses this problem.

Setting the BT_FUSE_SEL=0 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 the reset, the boot ROM code determines that the BT_FUSE_SEL is blown (BT_FUSE_SEL = 1) and the ROM code performs an internal boot according to the new eFUSE settings. This allows the user to set BOOT_MODE[1:0]=00b on a production device and burn the fuses on the same device (by forcing the entry to the Serial Downloader), without changing the value of the BOOT_MODE[1:0] or the pullups/pulldowns on the BOOT_MODE pins.

4.2.4 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 the boot code from the internal boot ROM.

The boot code performs the hardware initialization, loads the program image from the chosen boot device, performs the image validation using the HAB library (see Boot security settings), and then jumps to an address derived from the program image. If an error occurs during the 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 the Internal Boot, the boot flow may be controlled by a combination of eFUSE settings with an option of overriding the fuse settings using the General Purpose I/O (GPIO) pins. The GPIO Boot Select FUSE (BT_FUSE_SEL) determines whether the ROM uses the GPIO pins for a selected number of configuration parameters or eFUSEs in this mode.

- If BT_FUSE_SEL = 1, all boot options are controlled by the eFUSEs described in Table 4-2.
- If BT_FUSE_SEL = 0, the specific boot configuration parameters may be set using the 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 of the GPIO pins.

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

4.2.5 Boot security settings

The 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 the security hardware is initialized (the Security Controller or SNVS enters the Secure state), the DCD is processed if present, and the program image is authenticated by the HAB before its execution. All detected errors are logged, and the boot flow is aborted with the control being passed to the serial downloader. At this level, the execution does not leave the internal ROM unless the target executable image is 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. The security hardware is initialized (except for the SNVS which is left in the Non-Secure state), the DCD is processed if present, and the program image is authenticated by the HAB before its execution. All detected errors are logged, but have no influence on the boot flow which continues as if the errors did not occur. This configuration is useful for a secure product development because the program image runs even if the authentication data is missing or incorrect, and the error log can be examined to determine the cause of the authentication failure.
- Field Return: This level is intended for the parts returned from the shipped products.

4.3 Device configuration

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

This includes the boot device selection (SD, MMC, and so on), boot device configuration (SD bus width, speed, and so on), and other. In general, the source for this configuration comes from the eFUSEs embedded inside the chip. However, certain configuration parameters can be sourced from the GPIO pins, allowing further flexibility during the development process.

4.3.1 Boot eFUSE descriptions

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

Table 4-2. Boot eFUSE descriptions

Fuse Address	Configuration	Definition	GPIO[1]	Shipped value	Settings[2]
BT_FUSE_SEL 0x470[28]	OEM	In the 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 the GPIO pins or the eFUSE settings in the On-Chip OTP Controller (OCOTP). In the Boot From Fuse mode BOOT_MODE[1:0] = 00, the BT_FUSE_SEL fuse indicates whether the bit configuration eFuses are programmed.	NA	0	If BOOT_MODE[1:0] = 0b10: 0—The bits of the SBMR are overridden by the GPIO pins. 1—The specific bits of the SBMR are controlled by the eFUSE settings. If BOOT_MODE[1:0] = 0b00 0—The BOOT configuration eFuses are not programmed yet. The boot flow jumps to the serial downloader. 1—The BOOT configuration eFuses are programmed. The regular boot flow is performed.

Table continues on the next page...

Table 4-2. Boot eFUSE descriptions (continued)

SEC_CONFIG[1:0] 0 - 0x450[17] 1 - 0x470[25]	SEC_CONFIG [0] - NXP SEC_CONFIG [1] - OEM	Security Configuration, as defined in Boot security settings	NA	01	00—Reserved 01—Open (allows any program image, even if the authentication fails) 1x—Closed (The program image executes only if authenticated)
SRK_HASH[255:0] 0x580 - 0x5F0	OEM	256-bit hash value of the super root key (SRK_HASH)	NA	0	Settings vary—used by HAB
RECOVER_ECSPI_BOOT_EN 0x480[25]	OEM	ECSPI NOR recover boot enable bit	NA	0	0 - Disable 1 - Enable
SDP_DISABLE 0x480[21]	OEM	Serial download disable bit	NA	0	0 - Enable 1 - Disable
FORCE_BT_FROM_FUSE 0x480[20]	OEM	Force boot from fuse enable bit	NA	0	0 - Disable 1 - Enable. Read boot mode from boot mode fuses.
L1 I-Cache DISABLE 0x480[12]	OEM	L1 I Cache disable bit used by the boot during the entire execution.	No	0	0—L1 I Cache is enabled by the ROM during the boot. 1—L1 I Cache is disabled by the ROM during the boot.
DCACHE_DIS 0x480[8]	OEM	L1 Data Cache disable bit	NA	0	0 - Enable L1 D-Cache 1 - Disable L1 D-Cache
BT_FREQ 0x480[9]	OEM	Boot frequency selection	No	0	0—Arm—1000 MHz 1—Arm—500 MHz
LPB_BOOT 0x480[15:14]	OEM	USB Low-Power Boot	No	00	0x—LPB Disable 10—Divide by 2 11—Divide by 4
BT_LPB_POLARITY 0x480[13]	OEM	USB Low-Power Boot GPIO polarity	No	0	0—Low on the GPIO pad indicates the lowpower condition.

Table continues on the next page...

Table 4-2. Boot eFUSE descriptions (continued)

					1—High on the GPIO pad indicates the low-power condition.
WDOG_ENABLE 0x480[10]	OEM	Watchdog reset counter enable	No	0	0—The watchdog reset counter is disabled during the serial downloader. 1—The watchdog reset counter is enabled during the serial downloader.
WDOG_TIMEOUT_SELECT 0x480[17:16]	OEM	Watch Dog timeout value selection	NA	00	00 - 2s 01 - 1.5s 10 - 1s 11 - 0.5s
MMC_DLL_DLY[6:0] 0x490[14:8]	OEM	uSDHC Delay Line settings	No	0000000	uSDHC Delay Line settings
0x490[22:19]	OEM	Secondary boot image offset	NA	0	n == 0: Offset = 4MB n == 2: Offset = 1MB Others & n <= 10 : Offset = 1MB*2^n Secondary boot disables if n = fuse value bigger than 10. For FlexSPI NOR boot, the valid values are: 0, 1, 2, 3, 4, 5, 6, 7.
USDHC_PAD_SETTINGS 0x490[31:24] NAND_PAD_SETTIN GS 0x4A0[31:24]	OEM	Override values for the SD/MMC and NAND boot modes	No	00000000	Override the following IO PAD settings: [1:0] Driver Strength [2] Slew Rate [3] Hysteresis [4] Pull/Keeper select [6:5] Pull up/down config [7] Reserved.

1. This setting can be overridden by the GPIO settings when the BT_FUSE_SEL fuse is intact. See GPIO Boot Overrides for the corresponding GPIO pin.
2. 0 = intact fuse and 1= blown fuse

4.3.2 GPIO boot overrides

This table provides a list of the GPIO boot overrides:

Table 4-3. GPIO override contact assignments

Package pin	Direction on reset
BOOT_MODE0	Input
BOOT_MODE1	Input
BOOT_MODE2	Input
BOOT_MODE3	Input

The input pins provided are sampled at boot, and can be used to override the corresponding eFUSE values, depending on the setting of the BT_FUSE_SEL and FORCE_BT_FROM_FUSE fuse.

4.4 Device initialization

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

This includes details on:

- The ROM memory map
- The RAM memory map
- On-chip blocks that the ROM must use or change the POR register default values
- Clock initialization
- Enabling the MMU/L2 cache
- Exception handling and interrupt handling

4.4.1 Internal ROM/RAM memory map

These figures show the iROM memory map:

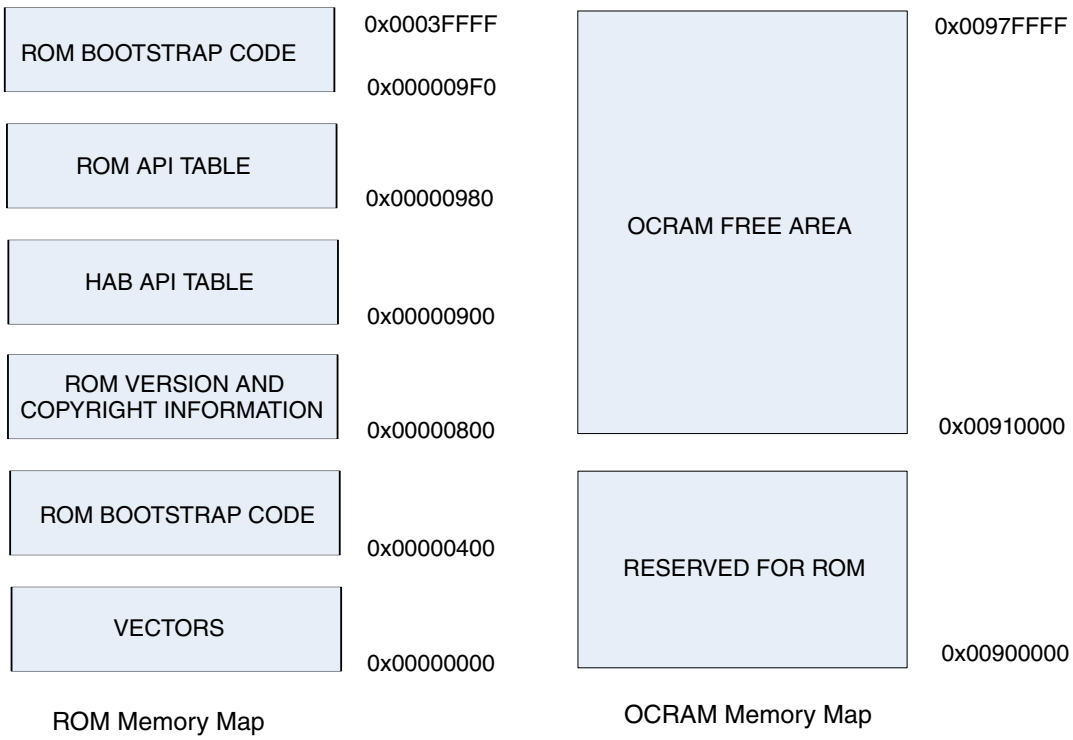


Figure 4-2. Internal ROM and RAM memory map

NOTE

If no ROM/HAB APIs are being used, the entire OCRAM region can be used freely after the 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 an alphabetical order) during the boot process. Note that the blocks actually used depend on the boot mode and the boot device selection:

- APBH—the DMA engine to drive the GPMI module
- BCH—62-bit error correction hardware engine with the AXI bus master and a private connection to the GPMI
- CCM—Clock Control Module
- ECSPI—Enhanced Configurable Serial Peripheral Interface
- FlexSPI—Flexible SPI Interface which supports serial NOR devices
- GPMI—NAND controller pin interface
- OCOTP_CTRL—On-Chip OTP Controller; the OCOTP contains the eFUSEs

- IOMUXC—I/O Multiplexer Control which allows the GPIO use to override the eFUSE boot settings;
- IOMUXC GPR—I/O Multiplexer Control General-Purpose Registers
- CAAM—Cryptographic Acceleration and Assurance Module
- SNVS—Secure Non-Volatile Storage
- SRC—System Reset Controller
- USB—used for the serial download of a boot device provisioning program
- USDHC—Ultra-Secure Digital Host Controller
- WDOG-1—Watchdog timer

4.4.3 Clocks at boot time

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

After the reset, each Arm core has access to all peripherals. The ROM code disables the clocks listed in the following table, except for the boot devices listed in the second column.

Table 4-4. PLL setting by ROM

PLL name	Frequency	Comment
ARM_PLL	1000 MHz	
SYS_PLL1	800 MHz	
SYS_PLL2	1000 MHz	

NOTE

All other PLLs are in the default status.

Table 4-5. Clock root setting by ROM

Clock Name	Frequency (MHz)	Source	Enable
ARM_A53_ROOT	1000	arm_pll_clk	Yes
ARM_M7_CLK_ROOT	200	system_pll2_200m_clk	
AHB_CLK_ROOT	133	system_pll1_133m_clk	Yes
MAIN_AXI_CLK_ROOT	333	system_pll2_333m_clk	Yes
NAND_CLK_ROOT	500	system_pll2_500m_clk	Enabled by driver
NAND_USDHC_BUS_CLK_ROOT	266	system_pll1_266m_clk	Enabled by driver
USB_BUS_CLK_ROOT		system_pll2_500m_clk	Enabled by driver
NOC_CLK_ROOT	400	system_pll1_800m_clk	Yes
USDHC1_CLK_ROOT	200	system_pll1_400m_clk	Enabled by driver
USDHC2_CLK_ROOT	200	system_pll1_400m_clk	Enabled by driver

Table continues on the next page...

Table 4-5. Clock root setting by ROM (continued)

Clock Name	Frequency (MHz)	Source	Enable
USDHC3_CLK_ROOT	200	system_pll1_400m_clk	Enabled by driver
USB_PHY_REF_CLK_ROOT		system_pll1_100m_clk	Enabled by driver
ECSPI1_CLK_ROOT	50	system_pll2_200m_clk	No
ECSPI2_CLK_ROOT	50	system_pll2_200m_clk	No
ECSPI3_CLK_ROOT	50	system_pll2_200m_clk	No
WRCLK_CLK_ROOT		system_pll1_40m_clk	No

NOTE

All other clock roots are in the default status.

Table 4-6. NAND_CLK_ROOT setting

NAND data rate	NAND_CLK_ROOT source	Frequency
Async/Legacy NAND	system_pll1_400m_clk	25 MHz
Sync 40M	system_pll1_400m_clk	40 MHz
Toggle/Sync 66M	system_pll1_400m_clk	66 MHz
Toggle 80M	system_pll1_400m_clk	80 MHz
Sync 100M	system_pll1_400m_clk	100 MHz
Toggle/Sync 133M	system_pll1_400m_clk	133 MHz
Sync 160M	system_pll1_400m_clk	133 MHz
Toggle/Sync 200M	system_pll1_400m_clk	200 MHz

NOTE

The NAND_CLK_ROOT source depends on the NAND data rate.

The ROM code disables the clocks listed in the following table, except for the boot devices listed in the "Enabled for boot device" column below.

Table 4-7. CCGR setting by ROM

Gating Register	LPCG Enable	Enabled for boot device
CCM_CCGR0	DVFS (GPC)	
CCM_CCGR1	Anamix	
CCM_CCGR2	CPU	
CCM_CCGR3	CSU	
CCM_CCGR4	Debug	
CCM_CCGR5	DDR1	
CCM_CCGR6	CM7 ATCLK	
CCM_CCGR7	ECSPI1	

Table continues on the next page...

Table 4-7. CCGR setting by ROM (continued)

Gating Register	LPCG Enable	Enabled for boot device
CCM_CCGR8	ECSPI2	
CCM_CCGR9	ECSPI3	
CCM_CCGR10	ENET1	
CCM_CCGR11	GPIO1	
CCM_CCGR12	GPIO2	
CCM_CCGR13	GPIO3	
CCM_CCGR14	GPIO4	
CCM_CCGR15	GPIO5	
CCM_CCGR16	GPT1	
CCM_CCGR17	GPT2	
CCM_CCGR18	GPT3	
CCM_CCGR19	GPT4	
CCM_CCGR20	GPT5	
CCM_CCGR21	GPT6	
CCM_CCGR22	HS	
CCM_CCGR23	I2C1	
CCM_CCGR24	I2C2	
CCM_CCGR25	I2C3	
CCM_CCGR26	I2C4	
CCM_CCGR27	IOMUX	
CCM_CCGR28	IOMUX1	
CCM_CCGR29	IOMUX2	
CCM_CCGR30	IOMUX3	
CCM_CCGR31	IOMUX4	
CCM_CCGR32	SNVSMIX	
CCM_CCGR33	MU	
CCM_CCGR34	OCOTP	
CCM_CCGR35	OCRAM	
CCM_CCGR36	OCRAM_s	
CCM_CCGR37	Reserved	
CCM_CCGR38	PERFMON1	
CCM_CCGR39	PERFMON2	
CCM_CCGR40	PWM1	
CCM_CCGR41	PWM2	
CCM_CCGR42	PWM3	
CCM_CCGR43	PWM4	
CCM_CCGR44	QoS	
CCM_CCGR45	QoS_Dispmix	
CCM_CCGR46	QoS_ENET	

Table continues on the next page...

Table 4-7. CCGR setting by ROM (continued)

Gating Register	LPCG Enable	Enabled for boot device
CCM_CCGR47	QSPI	
CCM_CCGR48	RAWNAND (APBHDMA, GPMI, BCH)	
CCM_CCGR49	RDC	
CCM_CCGR50	ROM	
CCM_CCGR51	SAI1	
CCM_CCGR52	SAI2	
CCM_CCGR53	SAI3	
CCM_CCGR54	SAI4	
CCM_CCGR55	SAI5	
CCM_CCGR56	SAI6	
CCM_CCGR57	SCTR	
CCM_CCGR58	SDMA1	
CCM_CCGR59	SDMA2	
CCM_CCGR60	SEC_DEBUG	
CCM_CCGR61	SEMA1	
CCM_CCGR62	SEMA2	
CCM_CCGR63	SIM_display	
CCM_CCGR64	SIM_ENET	
CCM_CCGR65	SIM_m	
CCM_CCGR66	SIM_main	
CCM_CCGR67	SIM_s	
CCM_CCGR68	SIM_wakeup	
CCM_CCGR69	SIM_HSIO	
CCM_CCGR70	Reserved	
CCM_CCGR71	SNVS	
CCM_CCGR72	Trace	
CCM_CCGR73	UART1	
CCM_CCGR74	UART2	
CCM_CCGR75	UART3	
CCM_CCGR76	UART4	
CCM_CCGR77	USB	
CCM_CCGR78	Reserved	
CCM_CCGR79	GPU3D	
CCM_CCGR80	Reserved	
CCM_CCGR81	USDHC1	
CCM_CCGR82	USDHC2	
CCM_CCGR83	WDOG1	
CCM_CCGR84	WDOG2	
CCM_CCGR85	WDOG3	

Table continues on the next page...

Table 4-7. CCGR setting by ROM (continued)

Gating Register	LPCG Enable	Enabled for boot device
CCM_CCGR86	Reserved	
CCM_CCGR87	GPU bus	
CCM_CCGR88	ASRC	
CCM_CCGR89	Reserved	
CCM_CCGR90	Reserved	
CCM_CCGR91	PDM	
CCM_CCGR92	GIC	
CCM_CCGR93	Display	
CCM_CCGR94	USDHC3	
CCM_CCGR95	SDMA3	
CCM_CCGR96	XTALOSC	
CCM_CCGR97	PLL	
CCM_CCGR98	TEMPSENSOR	
CCM_CCGR99	Reserved	
CCM_CCGR100	Reserved	
CCM_CCGR101	SAI7	
CCM_CCGR102	Reserved	

4.4.4 Enabling MMU and caches

The boot ROM includes a feature that enables the Memory Management Unit (MMU) and the caches to improve the boot speed.

The L1 instruction cache is enabled at the very beginning, unless the ICACHE_DISABLE fuse is blown. The MMU is always enabled and ROM enables it at the very beginning, after enabling the L1 ICACHE. The L1 data cache is enabled during image authentication and will be disabled after the image authentication is completed. The fuse, DCACHE_DIS, is used to control L1 DCACHE enable/disable. By default (fuse not programmed), ROM enables the L1 DCACHE.

4.4.5 Exception handling

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

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

After the 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 exception vector table in the RAM.

When an exception occurs, ROM does not move into USB serial download mode. Instead, ROM will set a flag (EXCEPTION_OCCURED) in the SRC GPR, and then reset the chip.

Mapping ROM Exception Vector Table

```
rom_vectors
B       startup                     ; offset is 0x000
ALIGN   0x80
B       default_exception_handler   ; offset is 0x080
ALIGN   0x80
B       default_exception_handler   ; offset is 0x100
ALIGN   0x80
B       default_exception_handler   ; offset is 0x180
ALIGN   0x80
B       sync_exception_handler      ; offset is 0x200
ALIGN   0x80
B       default_exception_handler   ; offset is 0x280
ALIGN   0x80
B       default_exception_handler   ; offset is 0x300
ALIGN   0x80
B       default_exception_handler   ; offset is 0x380
```

4.4.6 Interrupt handling during boot

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

4.4.7 Persistent bits

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

See this table for persistent bits list and description:

Table 4-8. Persistent bits

Bit name	Bit location	Description
-	SRC_GPR2[31:0]	Holds the FlexSPI NOR auto probe persistent content.
PERSIST_EXCEPTION_OCCURED	SRC_GPR10[17]	Exception cause reset

4.5 Boot devices (internal boot)

The chip supports these boot flash devices:

- Serial NOR flash via FlexSPI interface
- Raw NAND (MLC and SLC), and Toggle-mode NAND flash through GPMI-2 interface, located at CS0. Page sizes of 2 KB, 4 KB, and 8 KB. The bus widths of 8-bit with 2 through 62-bit BCH hardware ECC (Error Correction) are supported.
- SD/MMC/eSD/SDXC/eMMC4.4 via USDHC interface, supporting high capacity cards.

The selection of the external boot device type is controlled by BOOT_MODE[3:0]. See the table below for more details:

Table 4-9. Boot device selection

BOOT_MODE[3:0]	Boot device
0000	Boot from internal fuses
0001	USB Serial Download
0010	eMMC boot - by default, boots from USDHC3 port. Can be overridden by fuses settings.
0011	SD boot - by default, boots from USDHC2 port. Can be overridden by fuses settings.
0100	NAND 8-bit single device - 256 pages in block
0101	NAND 8-bit single device - 512 pages in block
0110	FlexSPI - 3B Read
0111	FlexSPI - Hyperflash 3.3V
1000	eCSPI Boot

4.5.1 Serial NOR Flash Boot via FlexSPI

4.5.1.1 Serial NOR eFUSE Configuration

Table 4-10. Fuse definition for Serial NOR over FlexSPI

Fuse Config	Config	Definitions	GPIO	Shipped Value	Settings
480[6:3]	OEM	xSPI FLASH Dummy Cycle	Yes	0	0 – Dummy cycles is auto-probed Others – Actual dummy cycles for Read command

Table continues on the next page...

Table 4-10. Fuse definition for Serial NOR over FlexSPI (continued)

Fuse Config	Config	Definitions	GPIO	Shipped Value	Settings
470[1:0]	OEM	xSPI FLASH Auto Probe Type	Yes	0	0 – QuadSPI NOR 1 – MXIC Octal 2 – Micron Octal 3 – Reserved
480[19:18]	OEM	Hold time before read from device	Yes	0	0 – 500us 1 – 1ms 2 – 3ms 3 – 10ms
470[2]	OEM	xSPI FLASH Auto Probe	Yes	0	0 – Disabled 1 – Enabled
BOOT_MODE[5:0]	OEM	FlexSPI interface selection	Yes	0	000110 – Flash with 3B READ 000111 – Hyperflash 3.3V
0x480[2:0]	OEM	xSPI FLASH Frequency	No	0	0 - 100 MHz 1 - 133 MHz 2 - 166 MHz 3 - 200 MHz 4 - 80 MHz 5 - 20 MHz Others – Reserved

NOTE

If the xSPI FLASH Auto Probe feature is enabled, the following is the logic how this feature works with other fuse combinations:

- Flash Type - If Flash type is 0, the "xSPI FLASH Auto Probe Type" takes effect for the Flash type selection. If Flash Type is greater than 1, the "Flash Type" Fuse is used for Flash type selection, ROM will issue specific command to probe the presence of Serial NOR FLASH.
- xSPI FLASH Frequency - This field is used for specifying the Flash working frequency.

4.5.1.2 FlexSPI Serial NOR Flash Boot Operation

The Boot ROM attempts to boot from Serial NOR flash if the BOOT_CFG [7:6] fuses are programmed to 0b'00 as shown in the Serial NOR eFUSE Configuration table, then the ROM will initialize FlexSPI1 interface. FlexSPI interface initialization is a two-step process.

The ROM expects the 512-byte FlexSPI NOR configuration parameters as explained in next section to be present at offset 0x400 in Serial NOR flash. The ROM reads these configuration parameters using the read command specified by BOOT_CFG [5:3] with Serial clock operating at 30 MHz.

In the second step, ROM configures FlexSPI1 interface with the parameters provided in configuration block read from Serial NOR flash and starts the boot procedure. Refer to Table 25-14 for details regarding FlexSPI configuration parameters and to the FlexSPI NOR boot flow chart for detailed boot flow chart of FlexSPI NOR.

Both booting an XIP and non XIP image are supported from Serial NOR Flash. For XIP boot, the image has to be built for FlexSPI address space and for non XIP the image can be built to execute from Internal RAM.

4.5.1.3 FlexSPI NOR boot flow chart

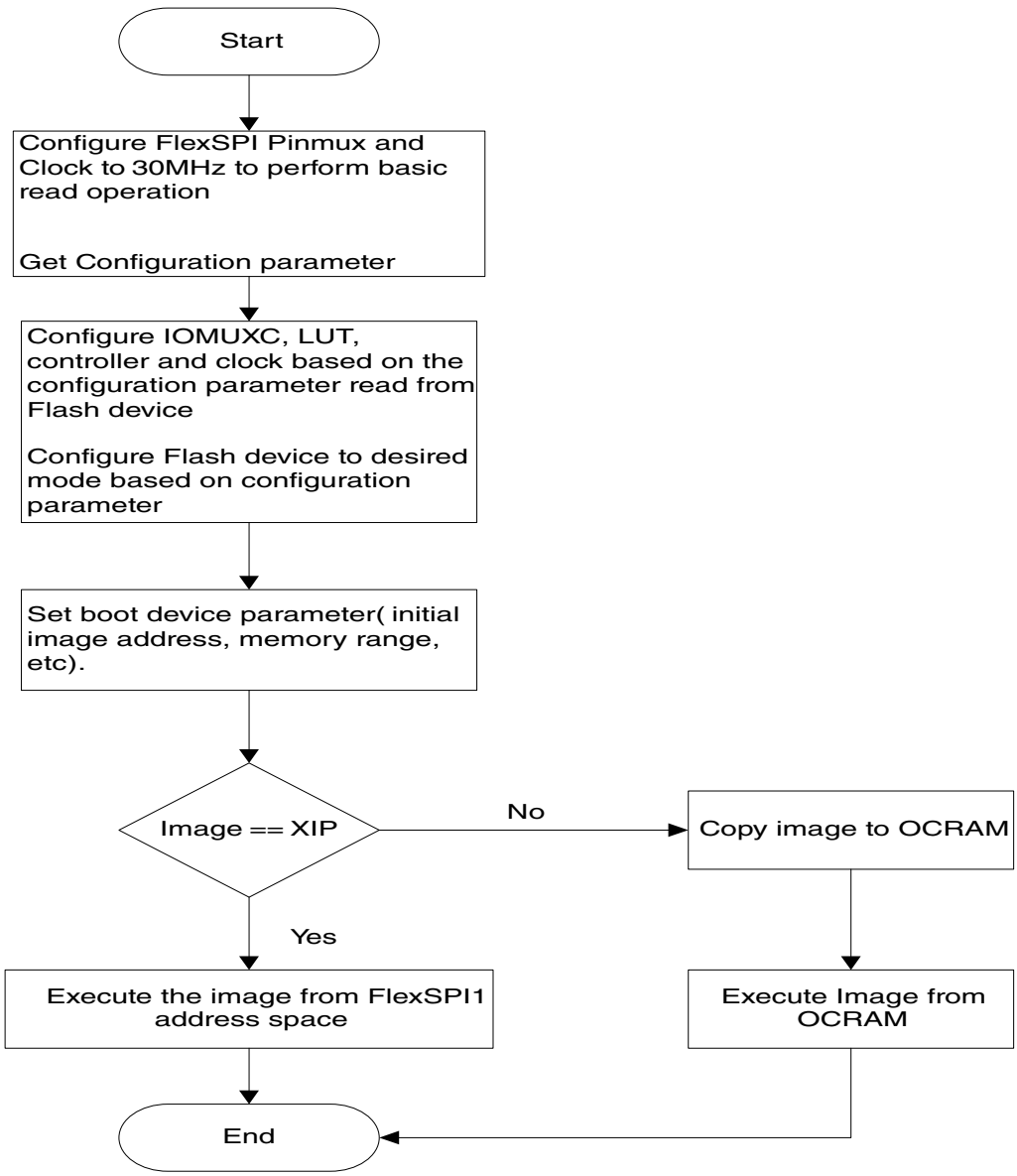


Figure 4-3. FlexSPI NOR boot flow

4.5.2 Serial NOR configuration based on FlexSPI interface

The ROM SW supports Serial NOR based on FlexSPI module, using a 448-bytes common FlexSPI configuration block and several specified parameters for Serial NOR respectively. See below sections for more details.

4.5.2.1 FlexSPI Configuration Block

FlexSPI Configuration block consists of parameters regarding specific Flash devices including read command sequence, quad mode enablement sequence (optional), etc.

Table 4-11. FlexSPI Configuration block

Name	Offset	Size(bytes)	Description
Tag	0x000	4	0x42464346, ascii:" FCFB"
Version	0x004	4	0x56010000 [07:00] bugfix = 0 [15:08] minor = 0 [23:16] major = 1 [31:24] ascii 'V'
-	0x008	4	Reserved
readSampleClkSrc	0x00C	1	0 – internal loopback 1 – loopback from DQS pad 3 – Flash provided DQS
dataHoldTime	0x00D	1	Serial Flash CS Hold Time Recommend default value is 0x03
dataSetupTime	0x00E	1	Serial Flash CS setup time Recommended default value is 0x03
columnAdressWidth	0x00F	1	3 – For HyperFlash 12/13 – For Serial NAND, see datasheet to find correct value 0 – Other devices
deviceModeCfgEnable	0x010	1	Device Mode Configuration Enable feature 0 – Disabled 1 – Enabled
-	0x011	3	Reserved
deviceModeSeq	0x014	4	Sequence parameter for device mode configuration
deviceModeArg	0x018	4	Device Mode argument, effective only when deviceModeCfgEnable = 1
configCmdEnable	0x01C	1	Config Command Enable feature 0 – Disabled 1 – Enabled
-	0x01D	3	Reserved
configCmdSeqs	0x020	16	Sequences for Config Command, allow 4 separate

Table continues on the next page...

Table 4-11. FlexSPI Configuration block (continued)

Name	Offset	Size(bytes)	Description
			configuration command sequences.
cfgCmdArgs	0x030	16	Arguments for each separate configuration command sequence.
controllerMiscOption	0x040	4	Bit0 – differential clock enable Bit1 – CK2 enable, must set to 0 in this silicon Bit2 – ParallelModeEnable, must set to 0 for this silicon Bit3 – wordAddressableEnable Bit4 – Safe Configuration Frequency enable set to 1 for the devices that support DDR Read instructions Bit5 – Pad Setting Override Enable Bit6 – DDR Mode Enable, set to 1 for device supports DDR read command
deviceType	0x044	1	1 – Serial NOR 2 – Serial NAND
sflashPadType	0x045	1	1 – Single pad 2 – Dual pads 4 – Quad pads 8 – Octal pads
serialClkFreq	0x046	1	Chip specific value, for this silicon 1 – 30 MHz 2 – 50 MHz 3 – 60 MHz 4 – 75 MHz 5 – 80 MHz 6 – 100 MHz 7 – 133 MHz 8 – 166 MHz Other value: 30 MHz
lutCustomSeqEnable	0x047	1	0 – Use pre-defined LUT sequence index and number 1 - Use LUT sequence parameters provided in this block

Table continues on the next page...

Table 4-11. FlexSPI Configuration block (continued)

Name	Offset	Size(bytes)	Description
-	0x048	8	Reserved
sflashA1Size	0x050	4	For SPI NOR, need to fill with actual size For SPI NAND, need to fill with actual size * 2
sflashA2Size	0x054	4	The same as above
sflashB1Size	0x058	4	The same as above
sflashB2Size	0x05C	4	The same as above
csPadSettingOverride	0x060	4	Set to 0 if it is not supported
sclkPadSettingOverride	0x064	4	Set to 0 if it is not supported
dataPadSettingOverride	0x068	4	Set to 0 if it is not supported
dqsPadSettingOverride	0x06C	4	Set to 0 if it is not supported
timeoutInMs	0x070	0	Maximum wait time during read busy status 0 – Disabled timeout checking feature Other value – Timeout if the wait time exceeds this value.
commandInterval	0x074	4	Unit: ns Currently, it is used for SPI NAND only at high frequency
dataValidTime	0x078	4	Time from clock edge to data valid edge, unit ns. This field is used when the FlexSPI Root clock is less than 100 MHz and the read sample clock source is device provided DQS signal without CK2 support. [31:16] data valid time for DLLB in terms of 0.1 ns [15:0] data valid time for DLLA in terms of 0.1 ns
busyOffset	0x07C	2	busy bit offset, valid range : 0-31
busyBitPolarity	0x07E	2	0 – busy bit is 1 if device is busy 1 – busy bit is 0 if device is busy
lookupTable	0x080	256	Lookup table
lutCustomSeq	0x180	48	Customized LUT sequence, see below table for details.
	0x1B0	16	Reserved for future use

Note:

1. To customize the LUT sequence for some specific device, users need to enable “lutCustomSeqEnable” and fill in corresponding “lutCustomSeq” field specified by command index below.
2. For Serial (SPI) NOR, the pre-defined LUT index is as follows:

Table 4-12. LUT sequence definition for Serial NOR

Command Index	Name	Index in lookup table	Description
0	Read	0	Read command Sequence
1	ReadStatus	1	Read Status command
2	WriteEnable	3	Write Enable command sequence
3	EraseSector	5	Erase Sector Command
4	PageProgram	9	Page Program Command
5	ChipErase	11	Full Chip Erase
6	Dummy	15	Dummy Command as needed
	Reserved	2,4,6,7,8,10,12,13,14	All reserved indexes can be freely used for other purpose

4.5.2.2 Serial NOR configuration block (512 bytes)

Table 4-13. Serial NOR configuration block

Name	Offset	Size (Bytes)	Description
memCfg	0	448	The common memory configuration block, see FlexSPI configuration block for more details
pageSize	0x1C0	4	Page size in terms of bytes, not used by ROM
sectorSize	0x1C4	4	Sector size in terms of bytes, not used by ROM
ipCmdSerialClkFreq	0x1C8	4	Chip specific value, not used by ROM For Ultra 0 – No change, keep current serial clock unchanged 1 – 30 MHz 2 – 50 MHz 3 – 60 MHz

Table continues on the next page...

Table 4-13. Serial NOR configuration block (continued)

Name	Offset	Size (Bytes)	Description
			4 – 75 MHz 5 – 80 MHz 6 – 100 MHz 7 – 133 MHz 8 – 166 MHz
Reserved	0x1CC	52	Reserved for future use

4.5.3 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 (BCH) is used to detect the errors.

4.5.3.1 NAND eFUSE configuration

The boot ROM determines the configuration of the external NAND flash by parameters, either provided by the eFUSE, or sampled on the GPIO pins during boot. See the Fusemap chapter in the Reference Manual for parameters details.

NOTE

BOOT_CFGx sampled on the GPIO pins depends on the BT_FUSE_SEL setting. See Boot Fusemap for details.

Table 4-14. NAND boot eFUSE descriptions

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
4A0[15]	OEM	BT_TOGGLE_MODE	Yes	0	0—raw NAND 1—toggle mode NAND
470[7:6]	OEM	Pages in block	Yes	0	00—32 pages 01—64 pages 10—128 pages 11—32 pages
4B0[11:10]	OEM	Row address cycles	Yes	00	00—3 01—2 10—4 11—5

Table continues on the next page...

Table 4-14. NAND boot eFUSE descriptions (continued)

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
4A0[12:9]	OEM	Toggle mode 33 MHz preamble delay, read latency	Yes	000	0000—16 GPMICLK cycles 0001—1 GPMICLK cycles 0010—2 GPMICLK cycles 0011—3 GPMICLK cycles 0100—4 GPMICLK cycles 0101—5 GPMICLK cycles 0110—6 GPMICLK cycles 0111—7 GPMICLK cycles 1000—8 GPMICLK cycles 1001—9 GPMICLK cycles 1010—10 GPMICLK cycles 1011—11 GPMICLK cycles 1100—12 GPMICLK cycles 1101—13 GPMICLK cycles 1110—14 GPMICLK cycles 1111—15 GPMICLK cycles
4A0[14:13]	OEM	Boot search count	Yes	00	00—2 01—2 10—4 11—8
0x4B0[7]	OEM	Override pad settings	No	0	Override the NAND pad settings 0—use the default values 1—use the PAD_SETTINGS value
0x4A0[31:24]	OEM	PAD_SETTINGS[7:0]	No	0	NAND pad settings value
0x4B0[15:12]	OEM	READ_RETRY_SEQ_ID[3:0]	No	0000	0000—don't use the ROM embedded read-retry sequence 0001—use Micron 20 nm read-retry sequence 0010—use Toshiba A19nm read-retry sequence 0011—use Toshiba 19nm read-retry sequence 0100—use SanDisk 19nm read-retry sequence 0101—use SanDisk 1ynm read-retry sequence 0110 to 1111—reserved

1. The setting can be overridden by the GPIO settings when the BT_FUSE_SEL fuse is intact. See Table 1 for the corresponding GPIO pin.

4.5.3.2 NAND flash boot flow and Boot Control Blocks (BCB)

There are two BCB data structures:

- FCB
- DBBT

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

The hardware ECC level to use is embedded inside the FCB block. The FCB data structure is also protected using the ECC. The driver reads raw 2112 bytes of the first sector and runs through the software ECC engine that determines whether the 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 the page number to the Search Stride number of pages to read for the next BCB until the SearchCount pages have been read.

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

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

This figure shows the state diagram of the FCB search:

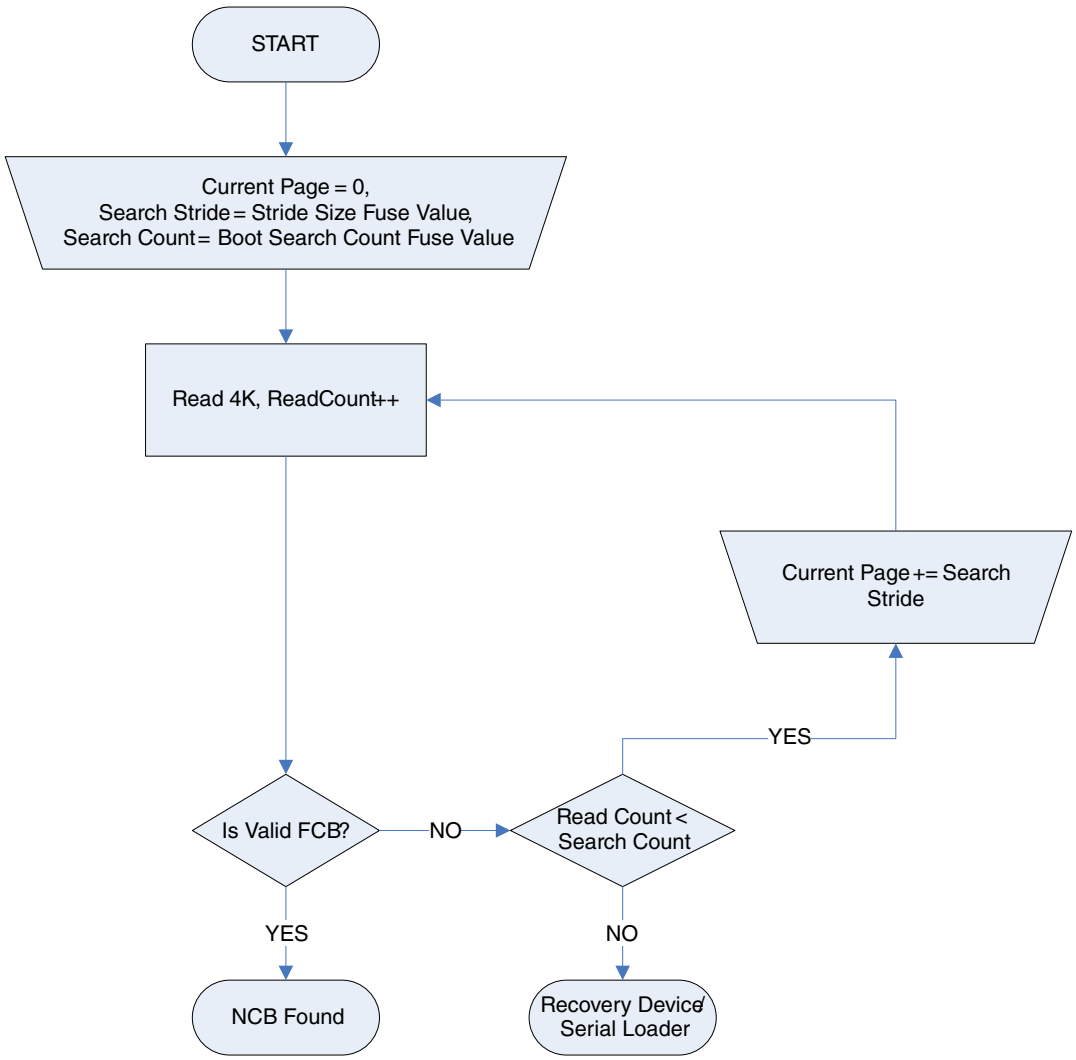


Figure 4-4. FCB search flow

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

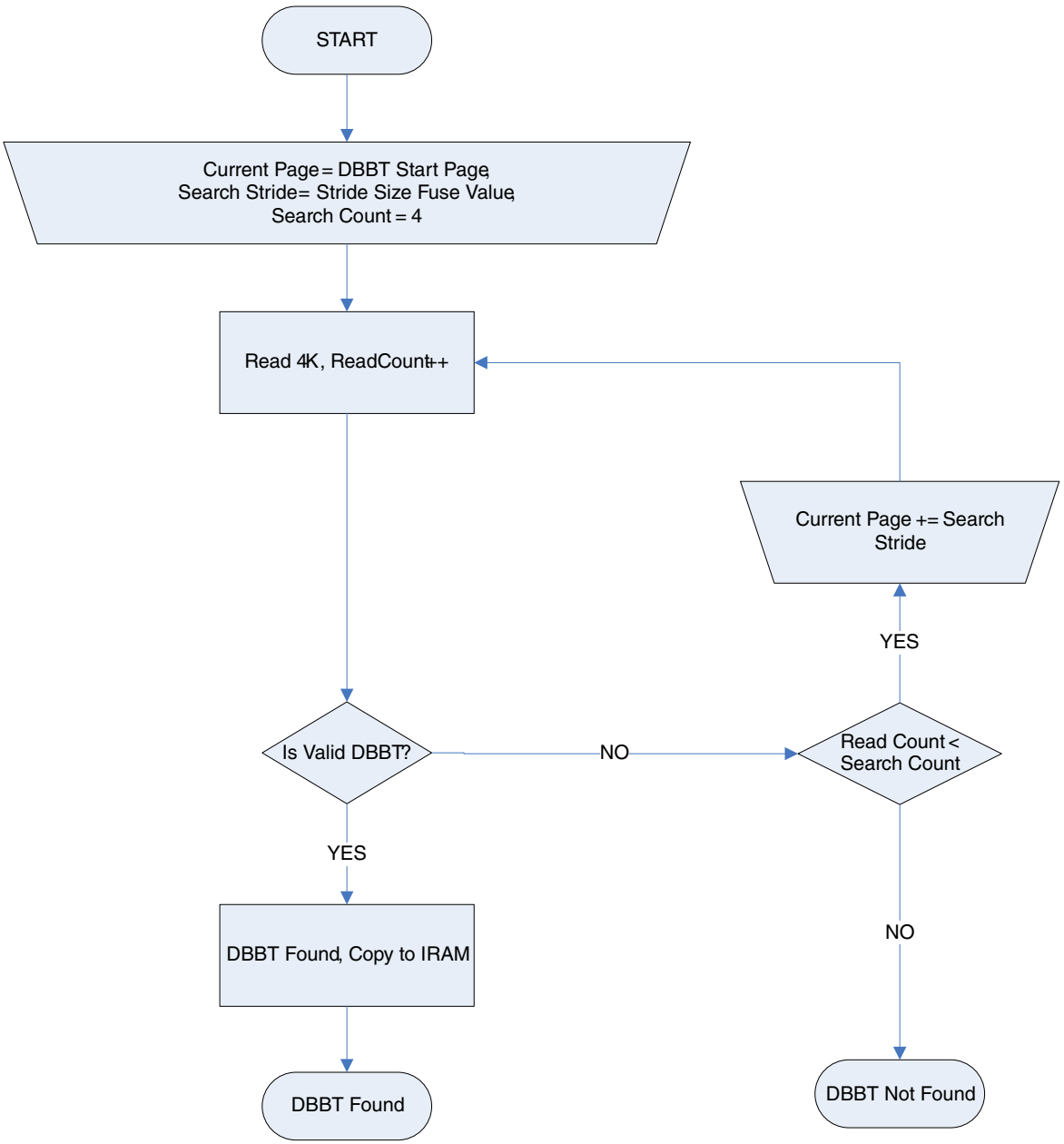


Figure 4-5. DBBT search flow

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 there is a page with a number of errors higher than ECC can correct during the primary image read, the boot ROM turns on the PERSIST_SECONDARY_BOOT bit and performs the software reset (After the software reset, the secondary image is used).

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

4.5.3.3 Firmware configuration block

The FCB is the first sector in the first good block. The FCB must 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 finds its copy in the next NAND block. The search area must span over at least two NAND blocks. The location information for the DBBT search area, FW1, and FW2 are all specified in the FCB. This table shows the flash control block structure:

Table 4-15. 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 0x20424346, in ascii "FCB"
Version	8	4	32-bit version number; this version of FCB is 0x00000001
m_NANDTiming	12	8	8 B of data for eight NAND timing parameters from the NAND datasheet. The eight 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. The ROM only uses the first four parameters, but the FCB provides space for other four parameters to be used by the bootloader or other applications.
PageDataSize	20	4	The 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	The total number of bytes in a page. Typically, 2112 for 2-KB page or 4224 or 4314 for 4-KB page or 8568 for 8-KB page.

Table continues on the next page...

Table 4-15. Flash control block structure (continued)

Name	Start byte	Size in bytes	Description
SectorsPerBlock	28	4	The number of pages per block. Typically 64 or 128 or depending on the NAND device type.
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 is used to set the BCH Error Corrrection level 0, 2, 4, .. or 62 for Block BN of ECC page, used in configuring the BCH62 page layout registers.
EccBlock0Size	48	4	Size of block B0 used in configuring the BCH62 page-layout registers.
EccBlockNSize	52	4	Size of block BN used in configuring the BCH62 page-layout registers.
EccBlock0EccType	56	4	Value from 0 to used to set the BCH Error Corrrection level 0, 2, 4, .. or 62 for Block BN of ECC page, used in configuring the BCH62 page layout registers.
MetadataBytes	60	4	Size of metadata bytes used in configuring the BCH62 page-layout registers.
NumEccBlocksPerPage	64	4	Number of the ECC blocks BN not including B0. This value is used in configuring the BCH62 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 the first copy of bootable firmware is located.
Firmware2_startingPage	108	4	Page number address where the second copy of bootable firmware is located.
PagesInFirmware1	112	4	Size of the first copy of firmware in pages.
PagesInFirmware2	116	4	Size of the second copy of firmware in pages.
DBBTSearchAreaStartAddress	120	4	Page address for the bad block table search area.
BadBlockMarkerByte	124	4	This is an input offset in the BCH page for the ROM to swap with the first byte of metadata after reading a page using the BCH62. The ROM supports the restoration of manufacturer-marked bad block markers in the page and this offset is the bad block marker offset location.

Table continues on the next page...

Table 4-15. Flash control block structure (continued)

Name	Start byte	Size in bytes	Description
BadBlockMarkerStartBit	128	4	This is an input bit offset in the BadBlockMarkerByte for the ROM to use when swapping eight bits with the first byte of metadata.
BBMarkerPhysicalOffset	132	4	This is the offset where the manufacturer leaves the bad block marker on a page.
BCHType	136	4	0 for BCH20 and 1 for BCH62. The chip is backwards compatible to BCH20 and this field tells the ROM to use the BCH20 or BCH62 block.
TMTiming2_ReadLatency	140	4	Toggle mode NAND timing parameter read latency, the ROM uses this value to configure the timing2 register of the GPMI.
TMTiming2_PreambleDelay	144	4	Toggle mode NAND timing parameter Preamble Delay. The ROM uses this value to configure the timing2 register of the GPMI.
TMTiming2_CEDelay	148	4	Toggle mode NAND timing parameter CE Delay. The ROM uses this value to configure the timing2 register of the GPMI.
TMTiming2_PostambleDelay	152	4	Toggle mode NAND timing parameter Postamble Delay. The ROM uses this value to configure the timing2 register of the GPMI.
TMTiming2_CmdAddPause	156	4	Toggle mode NAND timing parameter Cmd Add Pause. The ROM uses this value to configure the timing2 register of the GPMI.
TMTiming2_DataPause	160	4	Toggle mode NAND timing parameter Data Pause. The ROM uses this value to configure the timing2 register of the GPMI.
TMSpeed	164	4	This is the toggle mode speed for the ROM to configure the 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. The ROM uses this value to configure the timing1 register of the GPMI.
DISBBM	172	4	If 0, the ROM swaps the BadBlockMarkerByte with metadata[0] after reading a page using the BCH62. If the value is 1, the ROM does not swap.
BBMark_spare_offset	176	4	The offset in the metadata place which stores the data in the bad block marker place.
Onfi_sync_enable	180	4	Enable the Onfi nand sync mode support.
Onfi_sync_speed	184	4	Speed for the Onfi nand sync mode: 0 - 24 MHz, 1 - 33 MHz, 2 - 40 MHz, 3 - 50 MHz, 4 - 66 MHz, 5 - 80 MHz, 6 - 100 MHz, 7 - 133 MHz, 8 - 160 MHz, 9 - 200 MHz

Table continues on the next page...

Table 4-15. Flash control block structure (continued)

Name	Start byte	Size in bytes	Description
Onfi_syncNANDData	188	28	The parameters for the Onfi nand sync mode timing. They are read latency, ce_delay, preamble_delay, postamble_delay, cmdadd_pause, data_pause, and busy_timeout.
DISBB_Search	216	4	Disable the bad block search function when reading the firmware, only using DBBT.

The FCB data structure is protected using a 62-bit ECC. The layout of the FCB page is illustrated in this figure:

Figure 4-6. Layout of the FCB page

The detailed parameters of the FCB pages are listed in this table:

Table 4-16. Parameters setting for FCB page

Parameter	Value
TotalPageSize	2048+64=2112
MetadataBytes	32
EccBlock0Size	128
EccBlock0EccType	31
BCHType	0
EccBlockNSize	128
EccBlockNEccType	31
NumEccBlocksPerPage	7

To reduce the disturbances caused by a neighboring cell in the FCB page in the NAND chip, a randomizer is enabled when reading the FCB page. BCH ECC has a Randomizer module that is interfaced through the GPMI APBHDMA chain. The Randomizer can generate random data based on BCH ECC encoded/decoded data. It can be employed to reduce the disturbances caused by a neighboring cell in the NAND chip, thus reducing bit errors. The randomizer is used to reduce the bit errors in the FCB. Ensure that the randomizer is enabled when burning the FCB pages in the NAND flash. To control the

randomizer for the pages (except for FCB), a new field called Randomizer_Enable is added into the FCB structure. If the Randomizer_Enable field is set to 0, the randomizer is disabled. Reading the pages (except for FCB) being set to a non-zero value enables the randomizer. .

4.5.3.4 Discovered Bad Block Table (DBBT)

See this table for the DBBT format:

Table 4-17. 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 the 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

4.5.3.5 Bad block handling in ROM

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

To preserve the BI (bad block information), the flash updater or gang programmer applications must swap the Bad Block Information (BI) data to byte 0 of the metadata area for every page before programming the NAND flash. When the ROM loads the firmware, it copies back the value at metadata[0] to the BI offset in the page data. This figure shows how the factory bad block marker is preserved:

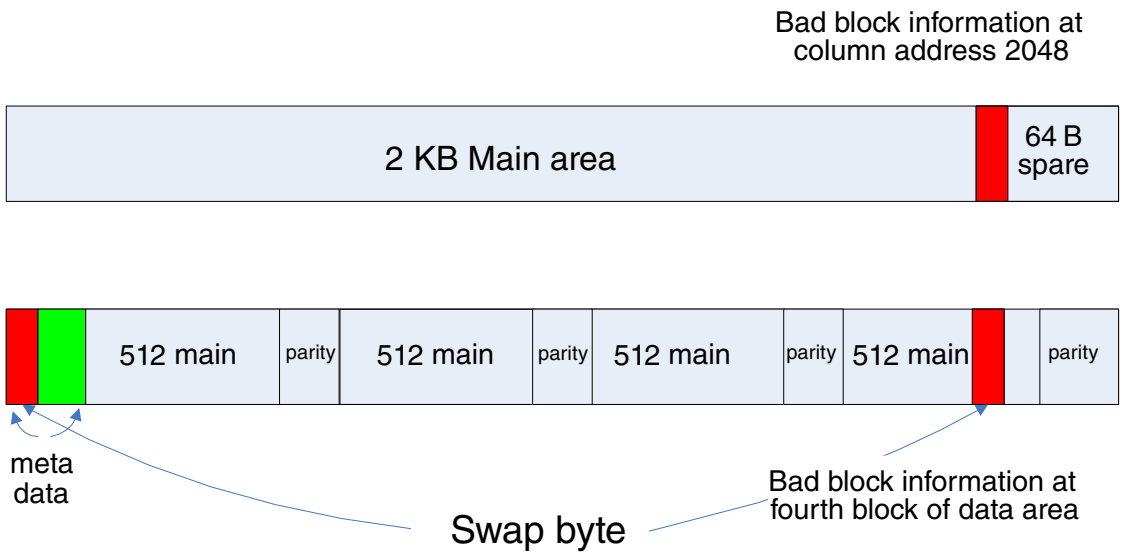


Figure 4-7. Factory bad block marker preservation

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 the manufacturer marked the bad block marker.

4.5.3.6 Toggle mode DDR NAND boot

If the BT_TOGGLEMODE efuse is blown, the ROM does the following to boot from the Samsung's toggle mode DDR NAND.

4.5.3.6.1 GPMI and BCH clocks configuration

The ROM sets the clock source and the dividers in the CCM registers.

If the BOOT_CFG is set (toggle mode), the GPMI/BCH CLK source is PLL2PFD4, and running at 66 MHz, otherwise the GPMI/ BCH CLK souce is PLL3, running at 24 MHz. The ROM sets the default values to timing0, timing1, and timing2 gpmi registers for 24 MHz clock speed. It uses the BOOT_CFG fuse to configure the GPMI timing2 register parameters preamble delay and read latency. The default value for these parameters is 2 when the fuses are not blown.

The default timing parameter values used by the ROM for the toggle-mode device are:

- Timing0.ADDRESS_SETUP = 5
- Timing0.DATA_SETUP = 10
- Timing0.DATA_HOLD = 10
- Timing1.DEVICE_BUSY_TIMEOUT = 0 x 500
- Timing2.READ_LATENCY = BOOT_CFG if blown, otherwise 2
- Timing2.CE_DELAY = 2
- Timing2.PREAMBLE_DELAY = BOOT_CFG if blown, otherwise 2
- Timing2.POSTAMBLE_DELAY = 3
- Timing2.CMDADD_PAUSE = 4
- Timing2.DATA_PAUSE = 6

The default timing parameters can be overriden by the TMTiming2_ReadLatency, TMTiming2_PreambleDelay, TMTiming2_CEDelay, TMTiming2_PostambleDelay, TMTiming2_CmdAddPause, and TMTiming2_DataPause parameters of the FCB.

4.5.3.6.2 Setup DMA for DDR transfers

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

4.5.3.6.3 Reconfigure timing and speed using values in FCB

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

The GPMI speed can be configured using the FCB parameter TMSpeed:

- 0—25 MHz
- 1—33 MHz
- 2—40 MHz
- 3—50 MHz
- 4—66 MHz
- 5—80 MHz
- 6—100 MHz
- 7—133 MHz
- 8—133 MHz
- 9—200 MHz

The GPMI timing0 register fields data_setup, data_hold, and address_setup are set to the values specified for the data_setup and data_hold and address_setup in the FCB member m_NANDTiming.

The GPMI timing1.DEVICE_BUSY_TIMEOUT is set to the value specified in the FCB member TMTiming1_BusyTimeout.

The GPMI timing2 register values are set using the FCB members TMTiming2.READ_LATENCY, CE_DELAY, PREAMBLE_DELAY, POSTAMBLE_DELAY, CMDADD_PAUSE, and DATA_PAUSE.

4.5.3.7 Typical NAND page organization

4.5.3.7.1 BCH ECC page organization

The first data block is called block 0 and the rest of the blocks are called block N. A separate ECC level scan is used for block 0 and block N.

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

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

This figure shows the valid layout for 2112-byte sized page.

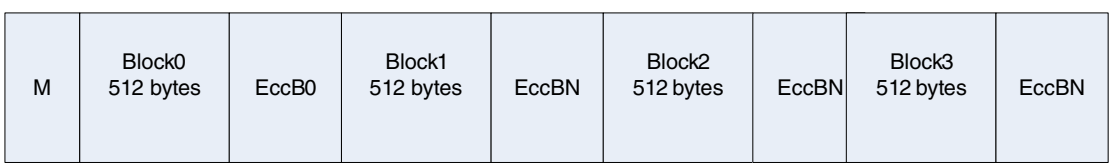


Figure 4-8. Valid layout for 2112-byte sized page

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

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

[M + (data_block_size x 4) + ([EccB0 + (EccBN x 3)] x 13) / 8] <= 2112;

M = metadata bytes and data_block_size is 512.

There are four data blocks of 512 bytes each in a page of 2-KB page sized NAND. The values of EccB0 and EccBN must be such that the above calculation does 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 4-9. Valid layout for 4-KB sized page

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

[M + (data_block_size x 8) + ([EccB0 + (EccBN x 7)] x 13) / 8] <= 4314;

M= metadata bytes and data_block_size is 512.

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

4.5.3.7.2 Metadata

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

4.5.3.8 IOMUX configuration for NAND

The following table shows the RawNAND IOMUX pin configuration.

Table 4-18. NAND IOMUX pin configuration

Signal	Pad name
NAND_ALE	NAND_ALE.alt0
NAND_CE0_B	NAND_CE0_B.alt0
NAND_CE1_B	NAND_CE1_B.alt0

Table continues on the next page...

Table 4-18. NAND IOMUX pin configuration (continued)

Signal	Pad name
NAND_CE2_B	NAND_CE2_B.alt0
NAND_CE3_B	NAND_CE3_B.alt0
NAND_CLE	NAND_CLE.alt0
NAND_DATA00	NAND_DATA00.alt0
NAND_DATA01	NAND_DATA01.alt0
NAND_DATA02	NAND_DATA02.alt0
NAND_DATA03	NAND_DATA03.alt0
NAND_DATA04	NAND_DATA04.alt0
NAND_DATA05	NAND_DATA05.alt0
NAND_DATA06	NAND_DATA06.alt0
NAND_DATA07	NAND_DATA07.alt0
NAND_DQS	NAND_DQS.alt0
NAND_RE_B	NAND_RE_B.alt0
NAND_READY_B	NAND_READY_B.alt0
NAND_WE_B	NAND_WE_B.alt0
NAND_WP_B	NAND_WP_B.alt0

4.5.4 Expansion device

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

4.5.4.1 Expansion device eFUSE configuration

The SD/MMC/eSD/eMMC/SDXC boot can be performed using either the USDHC ports, based on the setting of the BOOT_CFG fuses or it is associated to the GPIO input value at the boot.

All USDHC ports support the fast boot. See this table for details:

Table 4-19. USDHC boot eFUSE descriptions

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
BOOT_MODE	OEM	Boot device selection	Yes	0	0010 - eMMC boot, default from USDHC3 port 0011 - SD boot, default from USDHC2 port
0x470[10:9]	OEM	OVERRIDE_USDHC_BT_SEL_VAL	Yes	00	00 -- USDHC1 SD

Table continues on the next page...

Table 4-19. USDHC boot eFUSE descriptions (continued)

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
		The value to override USDHC BT CFG			01 -- USDHC1 EMMC 10 -- USDHC2 EMMC 11 -- USDCH3 SD
0x470[11]	OEM	OVERRIDE_USDHC_BT_SEL Enable to override the USDHC BT selection from Boot Mode	Yes	0	0 - Do not override 1 - Override
490[6]	OEM	Fast boot support	Yes	000	0 - Normal boot 1 - Fast boot
490[5:4]	OEM	Bus width	Yes	000	00 - 8-bit 01 - 4-bit 10 - 8-bit DDR (MMC 4.4) 11 - 4-bit DDR (MMC 4.4)
490[3:2]	OEM	Speed mode	Yes		SD speed selection 00 - Normal/SDR12 01 - High/SDR25 10 - SDR50 11 - SDR104 MMC speed selection 00 - Normal 01 - High
490[1]	OEM	USDHC IO Voltage Selection	Yes		USDHC1 IO VOLTAGE SELECTION (for Normal boot mode) 0 - 3.3 V 1 - 1.8 V
490[7]	OEM	SD power cycle enable/ eMMC reset enable	Yes	0	SD power cycle/eMMC reset 0 - Disabled 1 - Enabled
0x490[14:8]	OEM	SD/MMC DLL DLY config	No	0	Delay target for USDHC DLL, it is applied to the slave mode target delay or overrides the mode target delay, depending on the DLL override fuse bit value.
0x490[15]	OEM	USDHC DLL override enabled	No	0	0 - No override 1 - Override
0x490[16]	OEM	USDHC DLL enabled	No	0	0 - Disable the DLL for SD/eMMC 1 - Enable the DLL for SD/eMMC
0x490[18]	OEM	USDHC_IOMUX_SION_BIT_ENABLE	No	0	0 - Disable 1 - Enable

Table continues on the next page...

Table 4-19. USDHC boot eFUSE descriptions (continued)

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
0x490[23]	OEM	Disable SDMMC manufacture mode	No	0	0 - Enable 1 - Disable
0X490[31:24]	OEM	The low 8 bits of the value of the USDHC pad setting override	No	0	The low 8 bits of pad override settings
0x4A0[0]	OEM	Fast boot acknowledge enable	No	0	0 - Boot Ack disabled 1 - Boot Ack enabled
0x4A0[1]	OEM	The high 1 bit of the USDHC pad override settings.	NA	0	The High 1 bit of pad override settings.
0x4A0[2]	OEM	uSDHC power-off polarity selection	No	0	0 - Low 1 - High
0x4A0[3]	OEM	uSDHC power cycle delay selection	No	0	0 - 5 ms 1 - 2.5 ms
0x4A0[5:4]	OEM	uSDHC power cycle interval	No	0	00 - 20 ms 01 - 10 ms 10 - 5 ms 11 - 2.5 ms

1. The setting can be overridden by the GPIO settings when the BT_FUSE_SEL fuse is intact. See GPIO boot overrides for the corresponding GPIO pin.

The boot code supports these standards:

- MMCv4.4 or less
- eMMCv5.1 or less
- SDv2.0 or less
- eSDv2.10 rev-0.9, with or without FAST_BOOT
- SDXCv3.0

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

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

NOTE

The initial 4 KB of the program image must contain the IVT, DCD, and the Boot Data structures.

Table 4-20. 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 the application image length and the application destination pointer from the image.

4.5.4.2 MMC and eMMC boot

This table provides the MMC and eMMC boot details.

Table 4-21. MMC and eMMC boot details

Normal boot mode	During the initialization (normal boot mode), the MMC frequency is set to 347.22 KHz. When the MMC card enters the identification portion of the initialization, the voltage validation is performed, and the ROM boot code checks the high-voltage settings and the card capacity. The ROM boot code supports both the high-capacity and low-capacity MMC/eMMC cards. After the initialization phase is complete, the ROM boot code switches to a higher frequency (20 MHz in the normal boot mode or 40 MHz in the high-speed mode). The eMMC is also interfaced via the USDHC and follows the same flow as the MMC. 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 the BOOT_PARTITION_ENABLE field or the user partition was mentioned, the ROM boots from the user partition.
eMMC4.3 or eMMC4.4 device supporting special boot mode	If using an eMMC4.3 or eMMC4.4 device that supports the special boot mode, it can be initiated by pulling the CMD line low. If the BOOT ACK is enabled, the eMMC4.3/eMMC4.4 device sends the BOOT ACK via the DATA lines and the ROM can read the BOOT ACK [S010E] to identify the eMMC4.3/eMMC4.4 device. If the BOOT ACK is enabled, the ROM waits 50 ms to get the BOOT ACK and if the BOOT ACK is received by the ROM. If BOOT ACK is disabled ROM

Table continues on the next page...

Table 4-21. MMC and eMMC boot details (continued)

	waits 1 second for data. If the BOOT ACK or data was received, the eMMC4.3/eMMC4.4 is booted in the "boot mode", otherwise the eMMC4.3/eMMC4.4 boots as a normal MMC card from the selected boot partition. This boot mode can be selected by the BOOT_CFG (fast boot) fuse. The BOOT ACK is selected by the .
eMMC4.4 device	If using the eMMC4.4 device, the Double Data Rate (DDR) mode can be used. This mode can be selected by the BOOT_CFG2[7:5] (bus width) fuse.

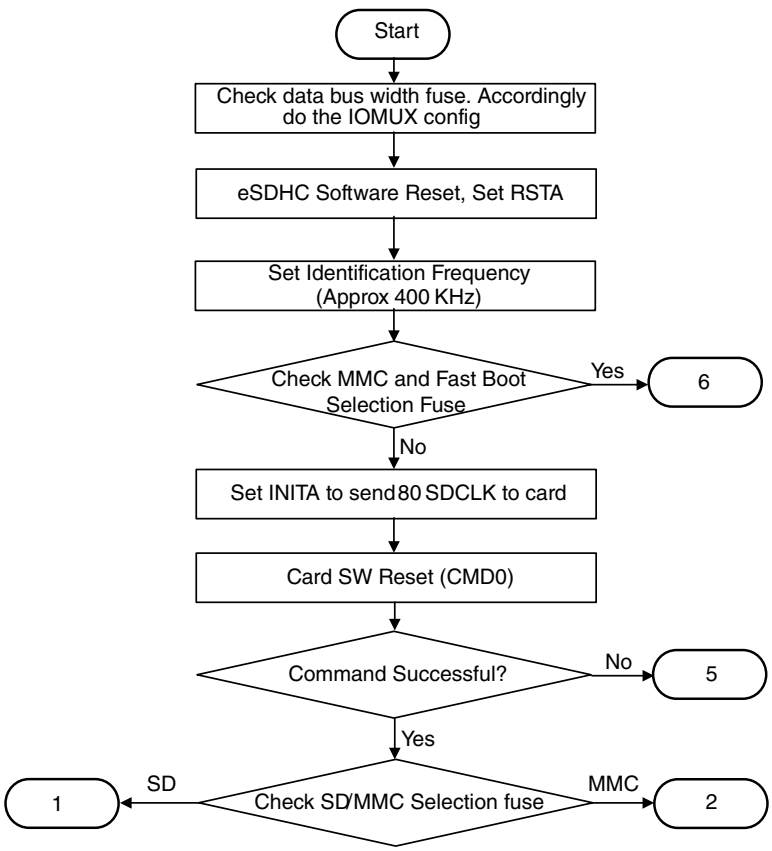


Figure 4-10. Expansion device boot flow (1 of 6)

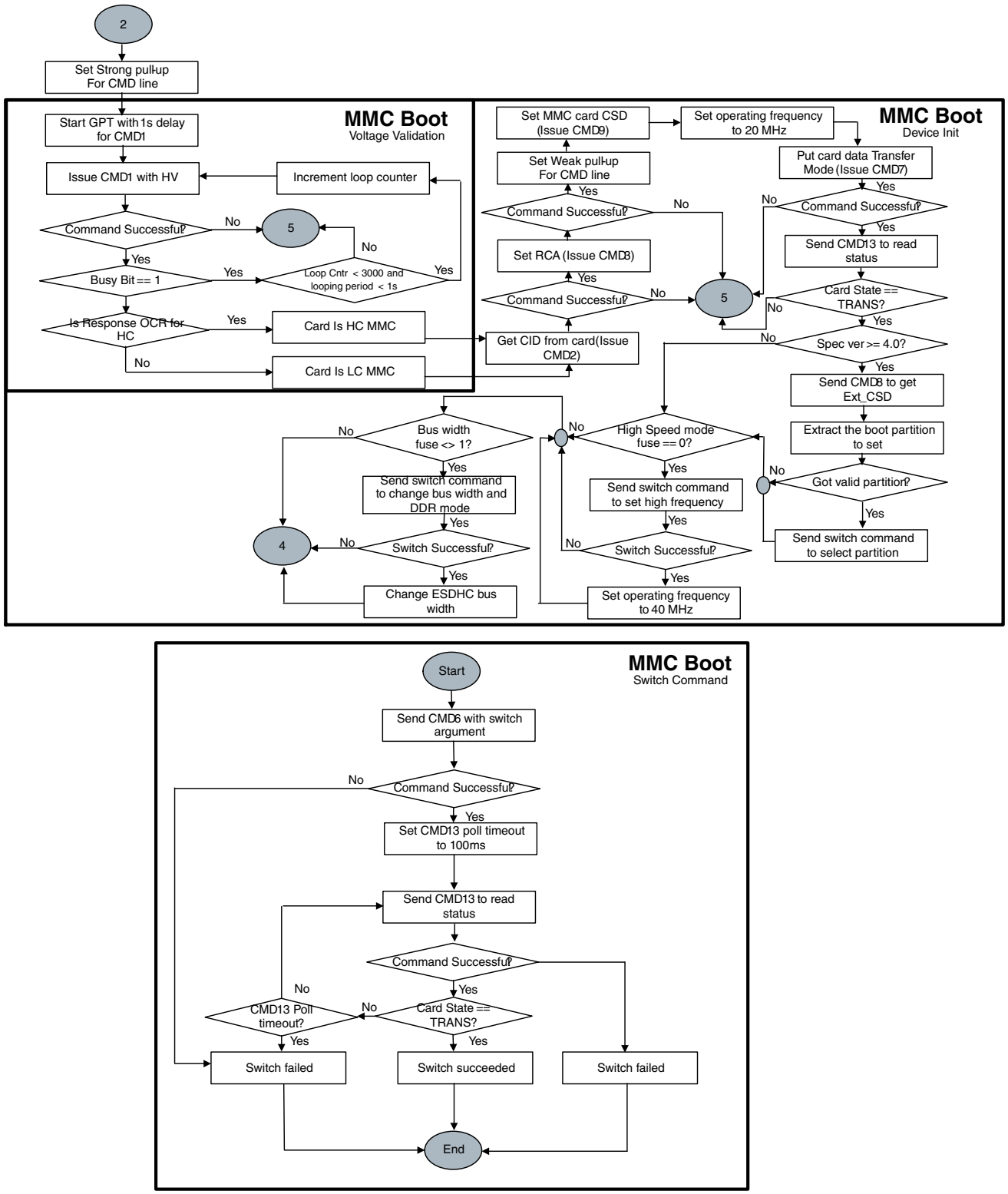


Figure 4-11. Expansion device (MMC) boot flow (2 of 6)

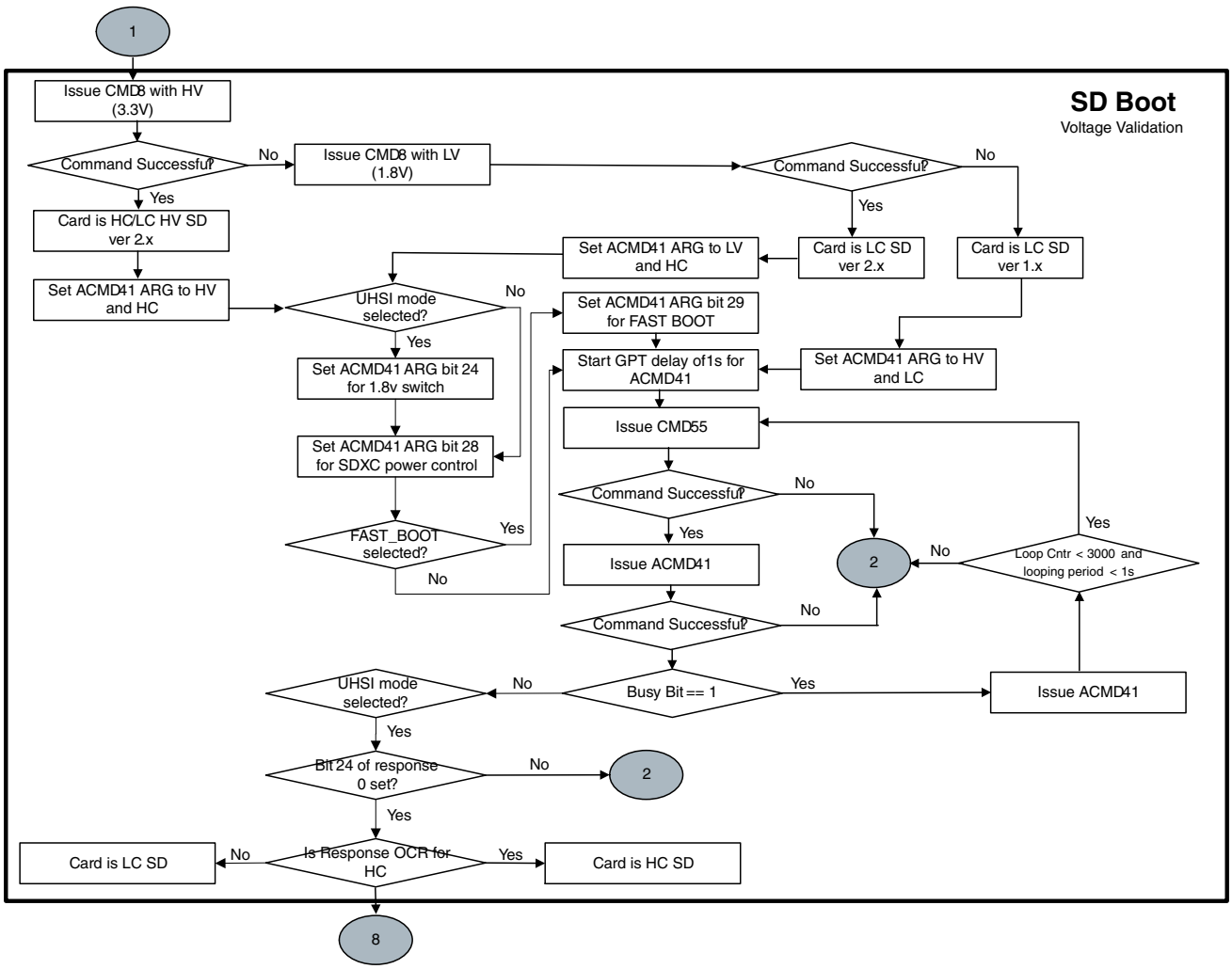


Figure 4-12. Expansion device (SD/eSD/SDXC) boot flow (3 of 6) part 1

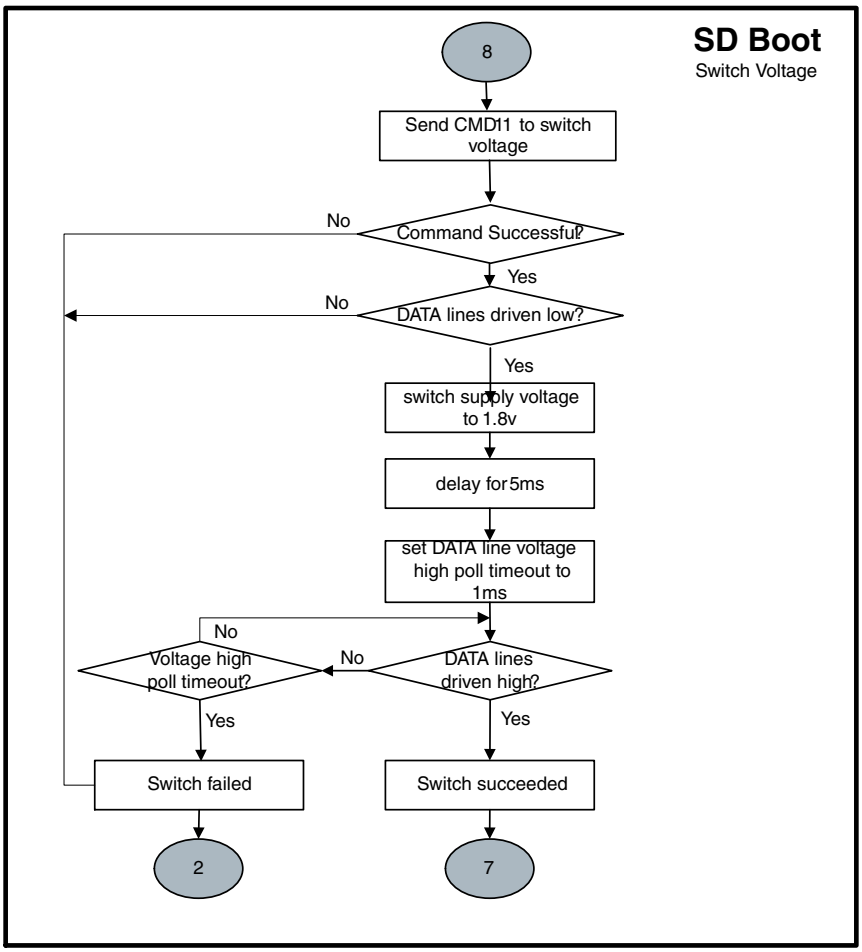


Figure 4-13. Expansion device (SD/eSD/SDXC) boot flow (3 of 6) part 2

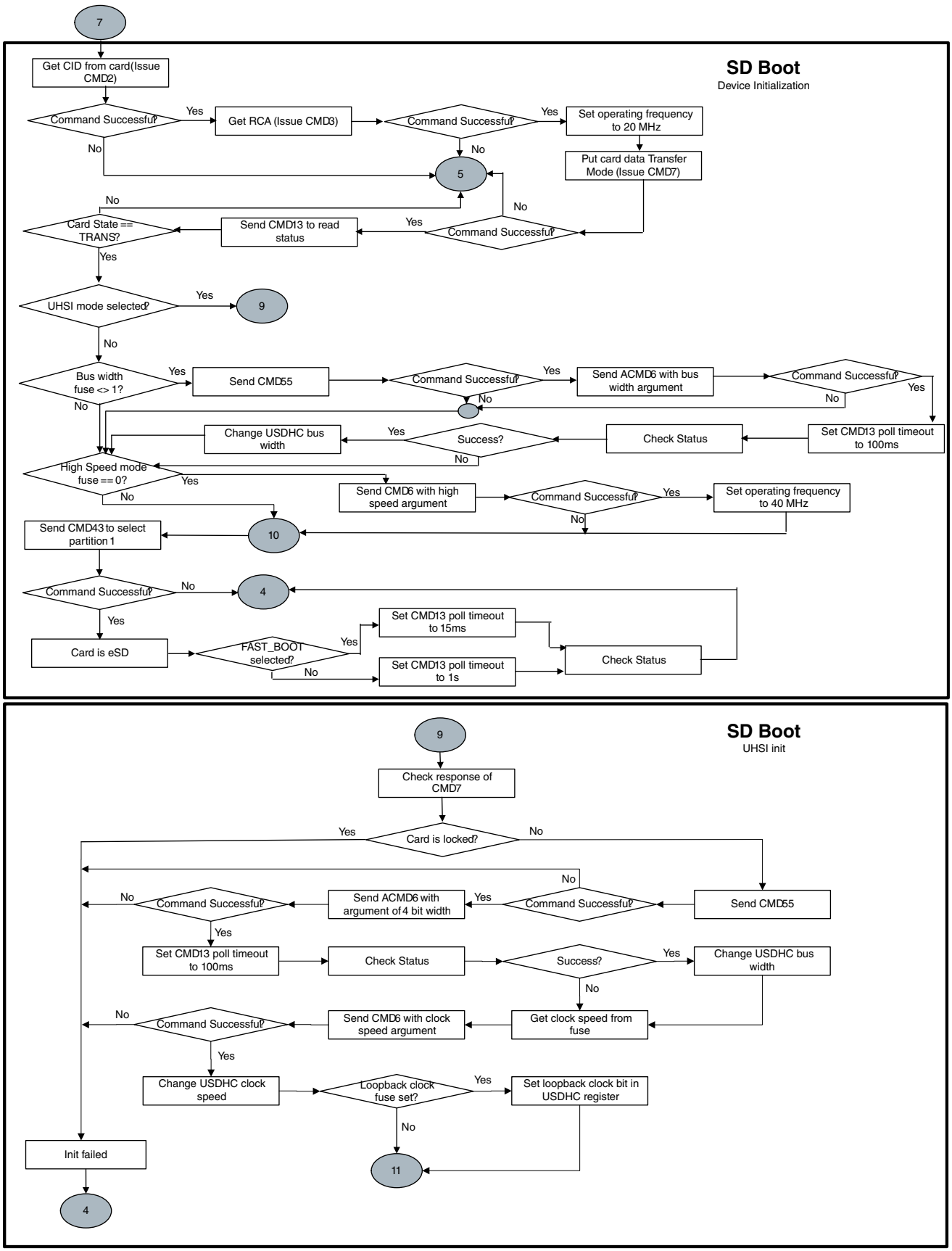


Figure 4-14. Expansion device (MMCSD/eSD/SDXC) boot flow (4 of 6)

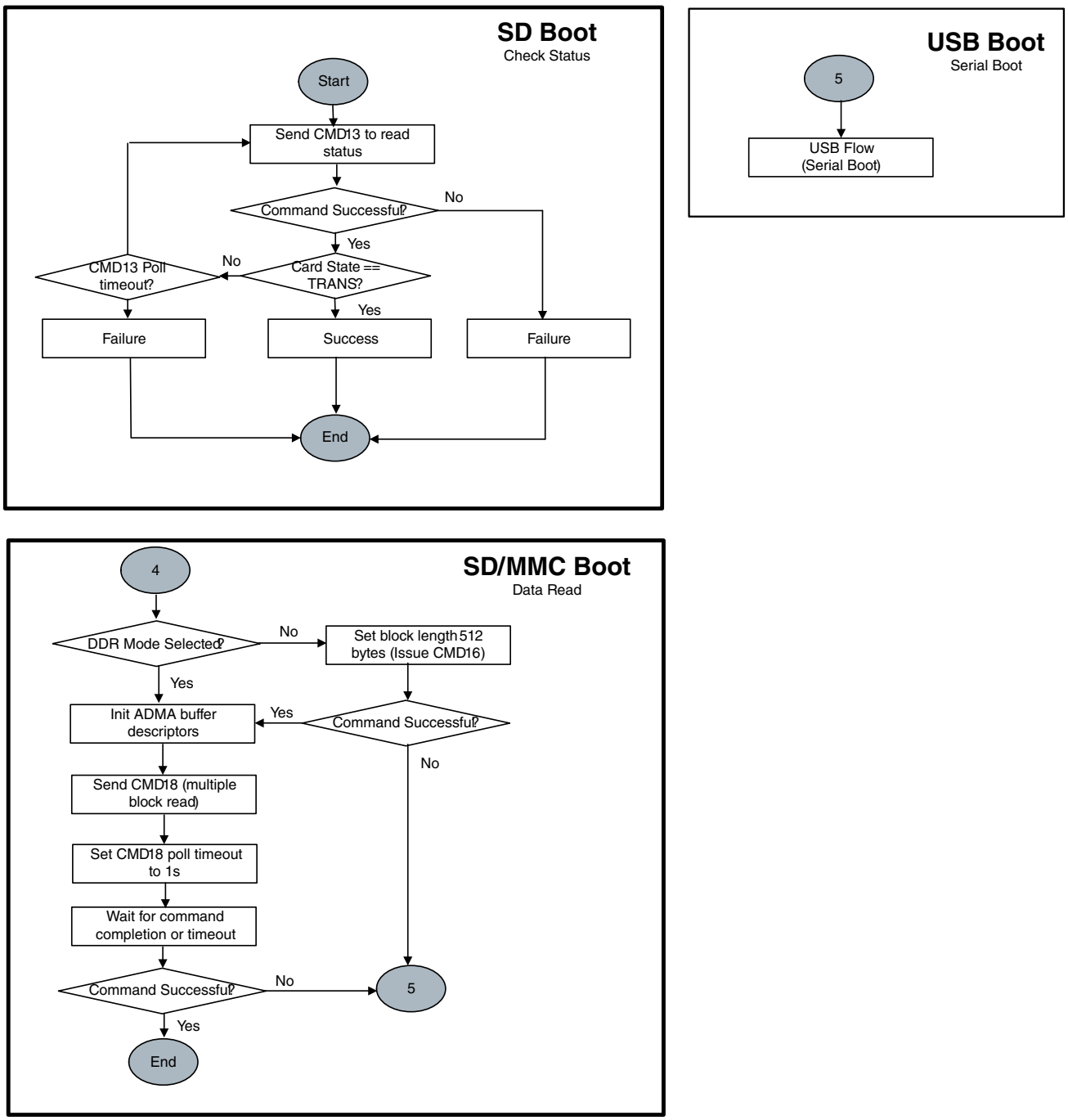


Figure 4-15. Expansion device (SD/eSD) boot flow (5 of 6)

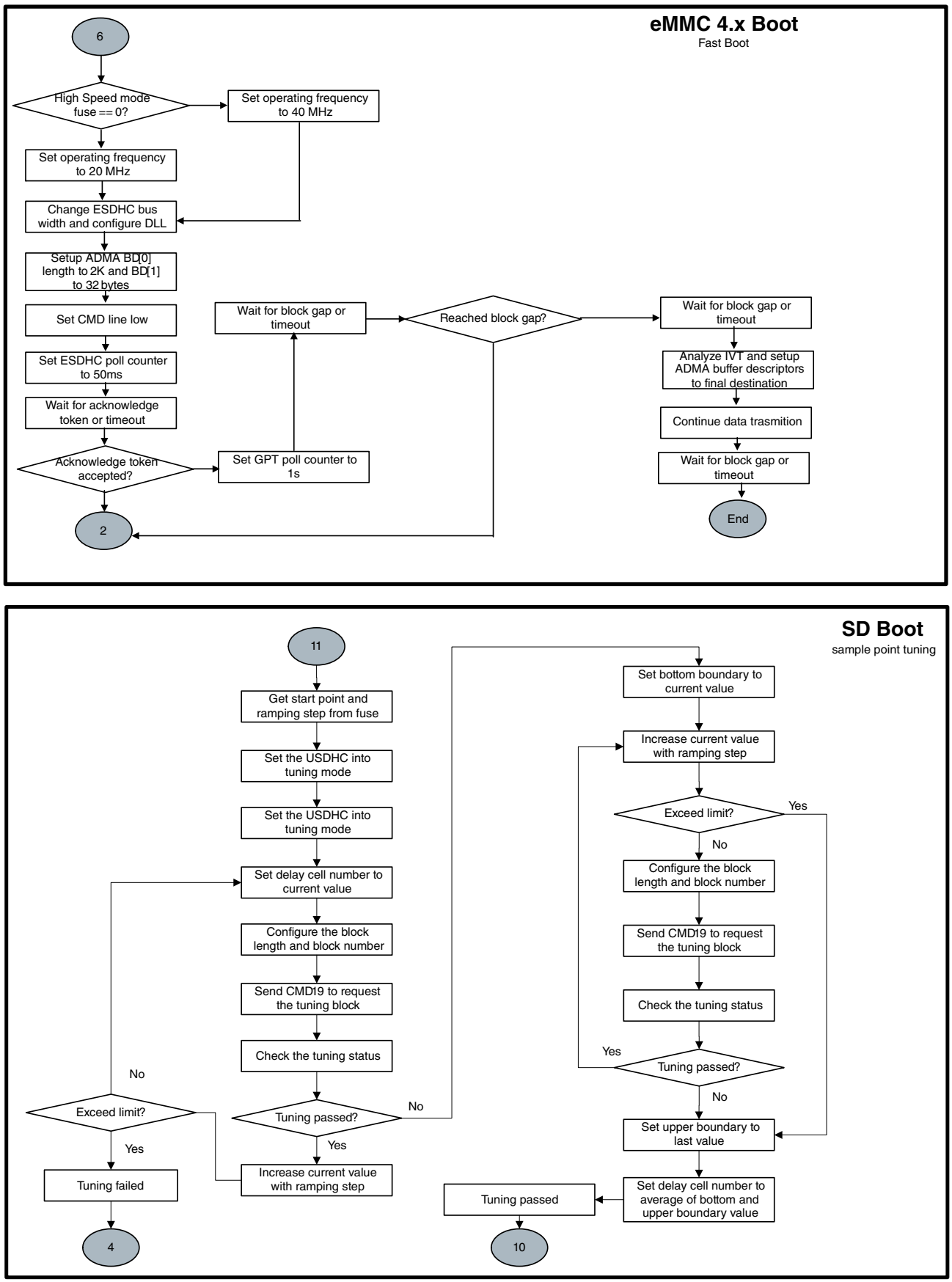


Figure 4-16. Expansion device boot flow (6 of 6)

4.5.4.3 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, the SD/eSD/SDXC card voltage validation is performed. During the voltage validation, the boot code first checks with the high-voltage settings; if that fails, it checks with the low-voltage settings.

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

During the 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 that the card is a normal SD or SDXC card. If it does not fail, the boot code assumes it is an eSD card. After the initialization phase is over, the boot code switches to a higher frequency (25 MHz in the normal-speed mode or 50 MHz in the high-speed mode). The ROM also supports the FAST_BOOT mode booting from the eSD card. This mode can be selected by the BOOT_CFG[4] (Fast Boot) fuse described in .

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

The UHSI calibration start value (MMC_DLL_DLY[6:0]) and the step value can be set to optimize the sample point tuning process.

If the SD Power Cycle Enable eFuse is 1, the ROM sets the SD_RST pad low, waits for 5 ms, and then sets the SD_RST pad high. If the SD_RST pad is connected to the SD power supply enable logic on board, it enables the power cycle of the SD card. This may be crucial in case the SD logic is in the 1.8 V states and must be reset to the 3.3 V states.

4.5.4.4 IOMUX configuration for SD/MMC

Table 4-22. SD/MMC IOMUX pin configuration

Signal	USDHC1	USDHC2	USDHC3
CLK	SD1_CLK.alt0	SD2_CLK.alt0	NAND_WE_B.alt2
CMD	SD1_CMD.alt0	SD2_CMD.alt0	NAND_WP_B.alt2
DATA0	SD1_DATA0.alt0	SD2_DATA0.alt0	NAND_DATA04.alt2

Table continues on the next page...

Table 4-22. SD/MMC IOMUX pin configuration (continued)

Signal	USDHC1	USDHC2	USDHC3
DATA1	SD1_DATA1.alt0	SD2_DATA1.alt0	NAND_DATA05.alt2
DATA2	SD1_DATA2.alt0	SD2_DATA2.alt0	NAND_DATA06.alt2
DATA3	SD1_DATA3.alt0	SD2_DATA3.alt0	NAND_DATA07.alt2
DATA4	SD1_DATA4.alt0	-	NAND_RE_B.alt2
DATA5	SD1_DATA5.alt0	-	NAND_CE2_B.alt2
DATA6	SD1_DATA6.alt0	-	NAND_CE3_B.alt2
DATA7	SD1_DATA7.alt0	-	NAND_CLE.alt2
RESET_B	SD1_RESET_B.alt5	SD2_RESET_B.alt5	GPIO1_IO09.alt4
VSELECT	GPIO1_IO03.alt1	GPIO1_IO04.alt1	GPIO1_IO11.alt4

4.5.5 Serial NOR through SPI

The chip supports booting from serial memory devices, such as EEPROM and serial flash, using the SPI.

These ports are available for serial boot: eCSPI (eCSPI1, eCSPI2, eCSPI3) interfaces.

4.5.5.1 Serial(SPI) NOR eFUSE configuration

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

See this table for details:

Table 4-23. Serial(SPI) NOR boot eFUSE descriptions

Fuse	Config	Definition	GPIO[1]	Shipped value	Settings
480[31:29]	OEM	ECSPI port selection	Yes	000	000 - eCSPI1 001 - eCSPI2 010 - eCSPI3
480[28]	OEM	SPI addressing	Yes	0	0 - 3 B (24-bit) 1 - 2 B (16-bit)
0x480[25]	OEM	Recovery boot enable	No	0	0 – Disabled 1 – Enabled
480[27:26]	OEM	CS selection (SPI only)	Yes	00	00 – CS#0

1. The setting can be overridden by the GPIO settings when the BT_FUSE_SEL fuse is intact. See Table 1 for the corresponding GPIO pin.

The ECPSI-1/ECPSI-2/ECPSI-3 block can be used as a boot device using the ECSPI interface for the serial(SPI) NOR boot. The SPI interface is configured to operate at 20 MHz for 3-byte addressing devices and at 4 MHz for 2-byte addressing devices.

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

NOTE

The Initial 4 KB of program image must contain the IVT, DCD, and the Boot Data Structures.

4.5.5.2 ECSPI boot

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

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

When using the SPI as a boot device, the chip supports booting from both the serial EEPROM and serial flash devices. The boot code determines which device is being used by reading the appropriate eFUSE/I/O values at the boot (see Table 4-23 for details).

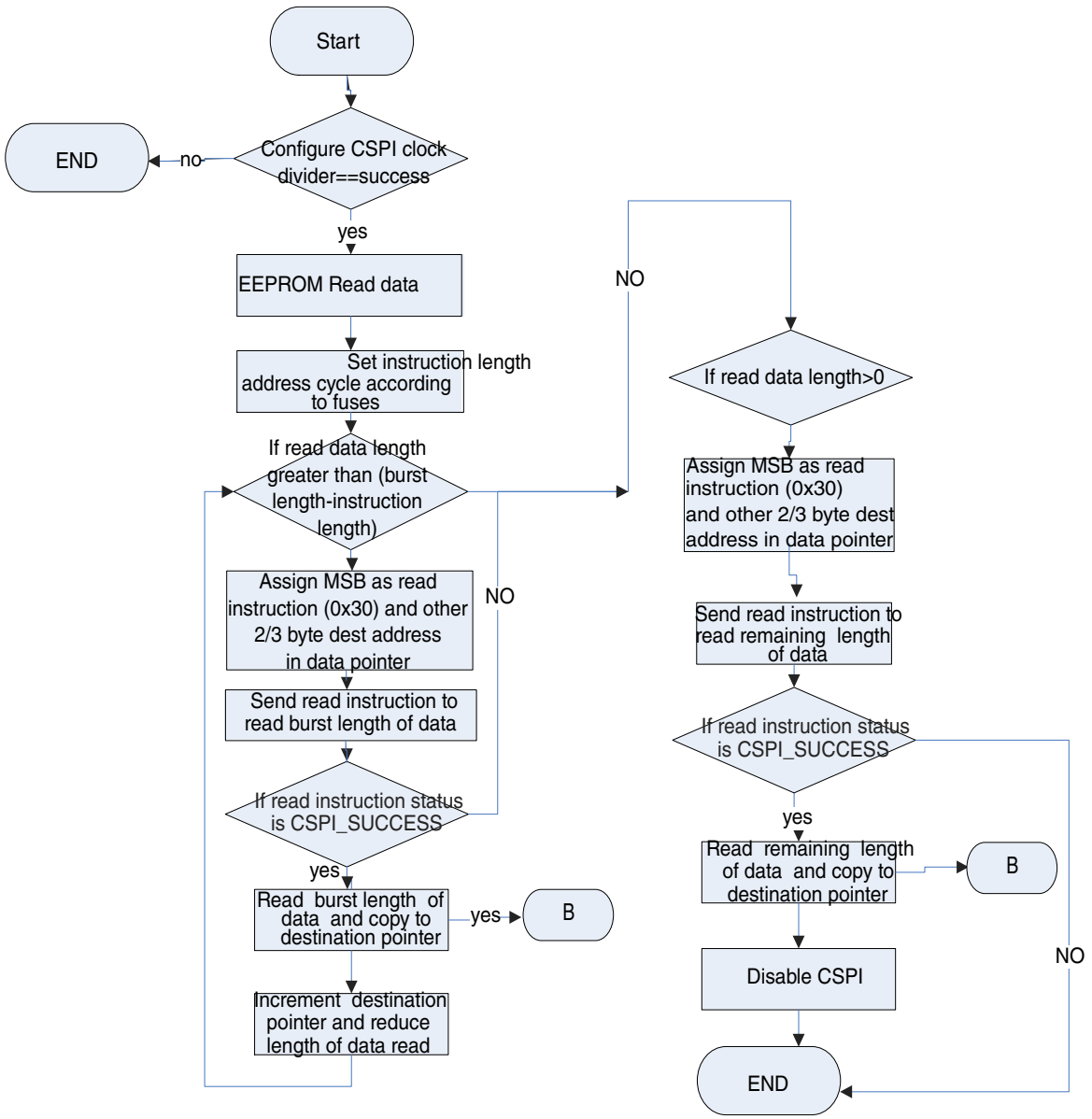


Figure 4-17. CSPI flow chart

4.5.5.2.1 ECSPI IOMUX pin configuration

The contacts assigned to the signals used by the CSPI blocks are shown in this table:

Table 4-24. ECSPI IOMUX pin configuration

Signal	eCSPI1	eCSPI2	eCSPI3
MISO	ECSPI1_MISO.alt0	ECSPI2_MISO.alt0	UART2_RXD.alt1
MOSI	ECSPI1_MOSI.alt0	ECSPI2_MOSI.alt0	UART1_TXD.alt1

Table continues on the next page...

Table 4-24. ECSPI IOMUX pin configuration (continued)

SCLK	ECSPI1_SCLK.alt0	ECSPI2_SCLK.alt0	UART1_RXD.alt1
SS0	ECSPI1_SS0.alt0	ECSPI2_SS0.alt0	UART2_TXD.alt1

4.6 Boot image

After the initial image (4KB) is loaded, ROM will check the image header's validity. The header must meet the following requirements:

- Reserved1 and Reserved2 of IVT must be 0.
- Device Configuration Data (DCD) of IVT must be 0.
- TAG of IVT header must be 0xD120004x
- SPL image's entry, boot_data, self, CSF in IVT need in IRAM_FREE_SPACE or FlexSPI space
- The address of the image's entry, boot data and CSF (if CSF is not NULL) must be out of IVT header range
- Image's boot_data must be in initial 4k image
- Image target address should be 4 bytes aligned, and the image need in IRAM_FREE_SPACE range or FlexSPI space. No plugin support.
- The first 4KB of IRAM_FREE_SPACE is used as an initial image buffer. If image target address is not equal to the base address of IRAM_FREE_SPACE, it should be out of the first 4KB range of IRAM_FREE_SPACE.

NOTE

The addresses referred in IVT header, boot data or CSF is from the boot core's view (A53 core).

4.6.1 Primary image offset and IVT offset

ROM supports one single image for all boot devices. The image offset and IVT offset for the supported boot devices is provided below.

Table 4-25. Primary image offset and IVT offset details

	Primary Image Offset	IVT Offset
SD	32KB (for GPT)	0
EMMC	0, if the image is in boot partition and 32K if it is in user partition	0

Table continues on the next page...

Table 4-25. Primary image offset and IVT offset details (continued)

	Primary Image Offset	IVT Offset
NAND	0	0
FlexSPI	4KB	0
SPI	0	0

NOTE

- Primary Image Offset - The image offset on device against to the beginning of boot device.
- IVT Offset - The IVT header offset in boot image against to the beginning of boot image.

4.6.2 Typical image placement in boot device

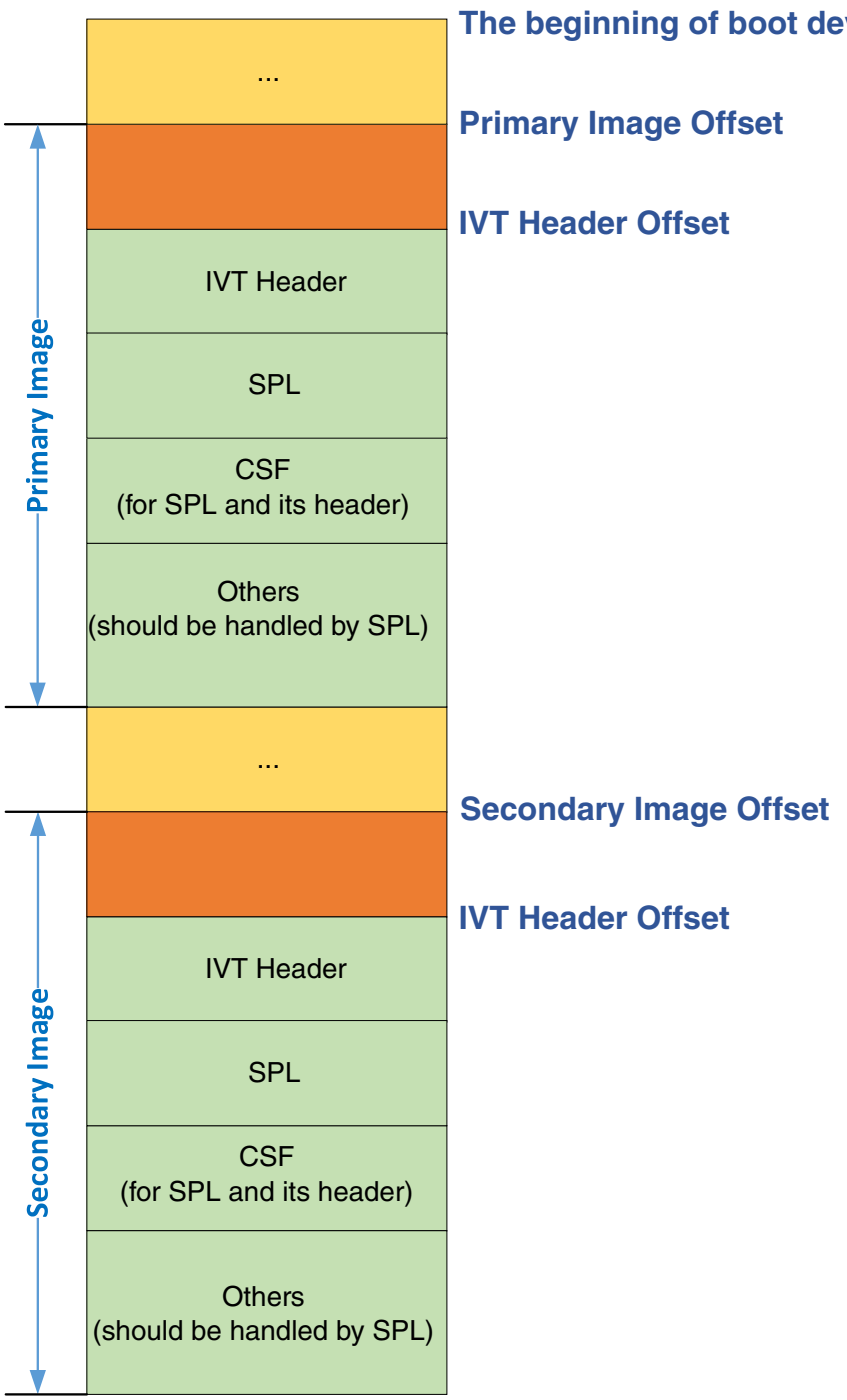


Figure 4-18. Typical image placement

- For the primary image offset and IVT offset details, please refer to Primary image offset and IVT offset.
- The secondary image offset is specified by fuses, please refer to the fuse map chapter.

4.7 USB boot

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

USB boot in ROM code works as a HID device. ROM takes USB as a normal boot device. The difference from the SD card here is that the USB is a 'stream' device, which is the same as eMMC fast boot. ROM must 'read' it one by one byte, and it can not specify the offset for the reading. This means that the image for USB boot must be continuous.

USB Serial Download boot can be disabled by fuse.

The USB HID device VID/PID and strings are listed in the following table:

Table 4-26. VID/PID and strings for USB HID device

Descriptor	Value
VID	0x1FC9
PID	0x012FU
bcdDevice	0x0001
String Descriptor0	bLength : 0x04 (4 bytes) bDescriptorType : 0x03 (String Descriptor) Language ID[0] : 0x0409 (English - United States)
String Descriptor1	bLength : 0x3A (58 bytes) bDescriptorType : 0x03 (String Descriptor) Language 0x0409 : "NXP SemiConductor Inc "
String Descriptor2	bLength : 0x1E (30 bytes) bDescriptorType : 0x03 (String Descriptor) Language 0x0409 : "SE Blank 815"
String Descriptor4	bLength : 0x22 (34 bytes) bDescriptorType : 0x03 (String Descriptor) Language 0x0409 : *This value depends on chip uuid.*

4.8 Low-power boot

The ROM supports the low-power boot. If the LPB_BOOT fuses are blown, the chip checks if there is a low-power condition via the pad. If there is a low-power boot condition, the ROM applies division factors on the ARM, DDR, AXI, and AHB root

clocks based on the LPB_BOOT fuses value (see the table below). The polarity of the low-power boot condition on the pad is set by the BT_LPB_POLARITY fuse (see the following figure).

Table 4-27. Low-power boot frequencies

LPB_BOOT	Boot Frequencies=0	Boot Frequencies=1
00	ARM_A53_CLK_ROOT= 1000 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 333 MHz	ARM_A53_CLK_ROOT= 500 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 166 MHz
01	ARM_A53_CLK_ROOT= 1000 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 333 MHz	ARM_A53_CLK_ROOT= 500 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 166 MHz
10	ARM_A53_CLK_ROOT= 500 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 166 MHz	ARM_A53_CLK_ROOT= 250 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 83.3 MHz
11	ARM_A53_CLK_ROOT= 250 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 83.3 MHz	ARM_A53_CLK_ROOT= 125 MHz AHB_CLK_ROOT= 133 MHz MAIN_AXI_CLK_ROOT= 41.67 MHz

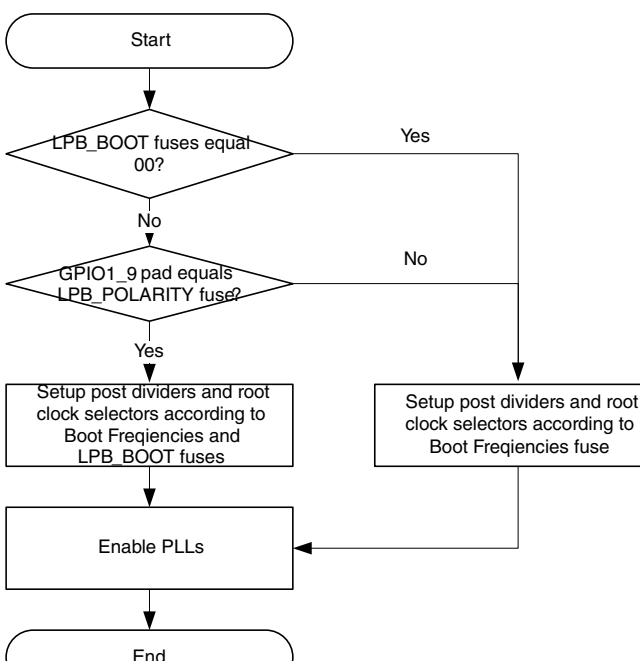


Figure 4-19. Low-power boot flow

4.9 SD/MMC manufacture mode

When the internal boot and recover boot (if enabled) failed, the boot goes to the SD/MMC manufacture mode before the serial download mode. In the manufacture mode, one bit bus width is used despite of the fuse setting.

By default, the SD/MMC manufacture mode is enabled. Blow the fuse of the DISABLE_SDMMC_MFG to disable it.

NOTE

A secondary boot is not supported in the SD/MMC manufacture mode.

Figure 4-20. SD/MMC manufacture boot flow

4.9.1 Using manufacture mode / serial download mode with eMMC

Manufacture mode is intended to allow a system to boot from a SD/MMC card on a board with unprogrammed boot media or to upgrade the image on a boot device. For manufacture mode, the boot ROM assumes if there is a SD/MMC card present indicated by the uSDHC card detect (CD) signal pulled low, then there is a valid image on the card.

If an unprogrammed eMMC device is connected to the uSDHC port(s) on which manufacture mode is supported and the CD signal is low, the ROM will attempt manufacturing mode with the eMMC device, which does not contain a valid image. The ROM loads the invalid image, and this will cause the ROM code to fail to enter serial download mode, resulting in a reset of the system by the ROM. To enter serial download mode in this case, the CD pin should be pulled up so the ROM does not detect a eMMC device present and will bypass the manufacture mode to enter serial download mode.

4.10 High-Assurance Boot (HAB)

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

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

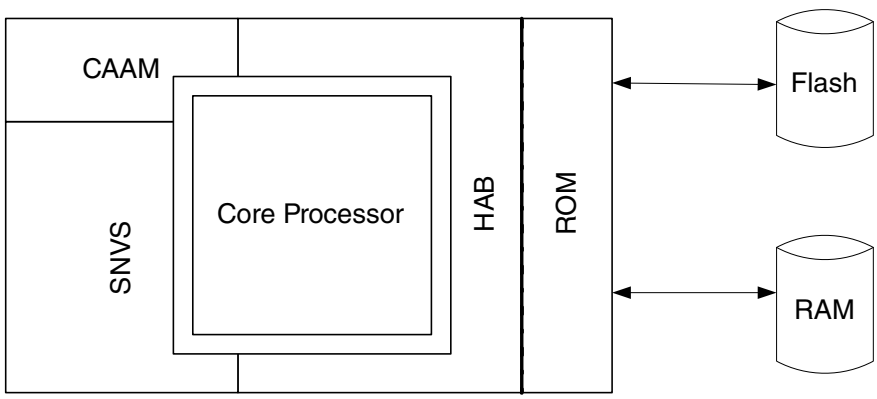


Figure 4-21. Secure boot components

The figure above illustrates the components used during a secure boot using HAB. The HAB interfaces with the SNVS to make sure that the system security state is as expected. The HAB also uses the CAAM hardware block to accelerate the SHA-256 message digest operations performed during the signature verifications and AES-128 operations for the encrypted boot operations. The HAB also includes a software implementation of SHA-256 for cases where a hardware accelerator can't be used. The RSA key sizes supported are 1024, 2048, 3072, and 4096 bits. The RSA signature verification operations are performed by a software implementation contained in the HAB library. The main features supported by the HAB are:

- X.509 public key certificate support
- CMS signature format support
- Proprietary encrypted boot support. Note that the encrypted boot depends on the CAAM hardware module. When the CAAM is disabled (when the EXPORT_CONTROL fuse is blown), the encrypted boot is not available.

NOTE

NXP provides the 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.nxp.com.

NOTE

For further details on using the secure boot feature using HAB, refer to *Secure Boot on i.MX 50, i.MX 53, i.MX 6 and i.MX 7 Series using HABv4 (AN4581).*

4.10.1 HAB API vector table addresses

For devices that perform a secure boot, the HAB library may be called by the boot stages that execute after the ROM code.

NOTE

For additional information on the secure boot including the HAB API, refer to *HABv4 RVT Guidelines and Recommendations (AN12263).*

4.11 Boot information for software

To address the requirement that the boot image may need to get the basic boot information when getting out of the boot process, the boot software information (Boot_SW_Info) is provisioned by the ROM.

The software must read the ROM address 0x9e8 to get the base address of the Boot_SW_Info data structure, and parse the Boot_SW_Info content to get the boot information.

Table 4-28. Boot_SW_Info structure

Offset	Byte3	Byte2	Byte1	Byte0
0x0	Reserved	Boot Device Type	Boot Device Instance	Reserved
0x4	Arm core frequency (in Hz)			
0x8	AXI bus frequency (in Hz)			
0xC	DDR frequency (in Hz)			
0x10	GPT1 input clock frequency (in Hz)			
0x14	Reserved			
0x18				
0x1C				

NOTE

The boot ROM sets the GPT1 in a free-running mode with a 32-kHz input clock.

Boot device type mapping:

- 0x1 - SD card or eSD chip
- 0x2 - MMC card or eMMC chip
- 0x3 - NAND chip
- 0x4 - FLEXSPI NOR
- 0x6 - ECSPI EEPROM

Boot device instance: The instance index of the boot device, starting from 0.

Chapter 5
Fusemap

5.1 Boot Fusemap

The boot mode utilizes 4 dedicated BOOT_MODE (BOOT_MODE[3:0]) pins and 2 functional pins (SAI3_TXD and SAI2_TXD0). The following section details the various modes and selection of the required boot devices.

Table 5-1. Boot Device Select

Boot Device Select	BOOT CODE	BOOT_MODE[5] (SAI3_TXD)	BOOT_MODE[4] (SAI2_TXD0)	BOOT_MODE[3]	BOOT_MODE[2]	BOOT_MODE[1]	BOOT_MODE[0]
Boot from internal fuses	0x00	0	0	0	0	0	0
USB Serial Download	0x01	0	0	0	0	0	1
USDHC3 (eMMC boot only, SD3 8-bit)	0x02	0	0	0	0	1	0
USDHC2 (SD boot only, SD2)	0x03	0	0	0	0	1	1
NAND 8-bit single device, 256 pages	0x04	0	0	0	1	0	0
NAND 8-bit single device, 512 pages	0x05	0	0	0	1	0	1
FlexSPI 3B Read	0x06	0	0	0	1	1	0
FlexSPI Hyperflash 3.3V	0x07	0	0	0	1	1	1
eCSPI Boot	0x08	0	0	1	0	0	0

NOTE

Fuses marked as “Reserved” are reserved for NXP internal (and future) use only. Customers should not attempt to burn these, as the IC behavior may be unpredictable. The reserved fuses can be read as either 0 or 1.

Table 5-2. Boot Fusemap

Addr	7	6	5	4	3	2	1	0
0x470[7:0]	OVERRIDE_NAND_PG_PER_BLK_VAL 00 - 32 pages 01 - 64 pages 10 - 128 pages 11 - 32 pages		OVERRIDE_FLEXSPI_BT_SEL 0 - Do not override 1 - Override	OVERRIDE_FLEXSPI_BT_SEL_VAL 00 - FlexSPI (Hyperflash 1.8V) 01 - FlexSPI (Flash with 4B READ (1x13 default supported)) 10 - Default Octal mode (Micron) 11 - Default Octal mode (Mxic)		FLEXSPI_AUTO_PROBE_EN 0 - Disable 1 - Enable	FLEXSPI_AUTO_PROBE_TYPE 00 - QuadSPI NOR 01 - MxicOctal 10 - MicronOctal 11 - AdestoOctal	
0x470[15:8]	BOOT_MODE_FUSES Boot Rom will retrieve boot mode from these fuses instead of BOOT_MODE pins if: * BOOT_MODE_PINS=0x0 or * BT_FUSE_SEL blown				OVERRIDE_USDHC_BT_SEL 0 - Do not override 1 - Override	OVERRIDE_USDHC_BT_SEL_VAL 00 - uSDHC1 SD 01 - uSDHC1 eMMC 10 - uSDHC2 eMMC 11 - uSDHC3 SD		OVERRIDE_NAND_PG_PER_BLK 0 - Do not override 1 - Override
0x480[7:0]	Reserved	FLEXSPI_DUMMY_CYCLE_SEL				FLEXSPI_FEQ_SEL 000 - 100 MHz 001 - 133 MHz 010 - 166 MHz 011 - 200 MHz 100 - 80 MHz 101 - 20 MHz		
0x480[15:8]	BT_LPB (Core/DDR/Bus) '00'/'01' - LPB Disable '10' - Div by 2 '11' - Div by 4		BT_LPB_POLARITY (GPIO polarity)	ICACHE_DIS L1 I-Cache DISABLE	TZASC_EN	WDOG_EN '0' - Disabled '1' - Enabled	BT_FREQ_SEL (ARM/DDR) 0 - 800 / 800 MHz 1 - 400 / 400 MHz	DCACHE_DIS
0x480[23:16]	NOC_ID_REMAP_BYPASS	ROM_NO_LOG If blown, ROM will not log event to log buffer.	SDP_DISABLE Disable USB serial download	FORCE_BT_FROM_FUSE Boot from programmed fuses, not Boot Mode Pins	FLEXSPI_HOLD_TIME_SEL 00 - 500us 01 - 1ms 10 - 3ms 11 - 10ms		WDOG_TIMEOUT_SELECT 00 - 2.0s 01 - 1.5s 10 - 1.0s 11 - 0.5s	
0x480[31:24]	ECSPI_PORT_SEL 000 - eCSPI1 001 - eCSPI2 010 - eCSPI3			ECSPI_ADDR_SEL 0 - 3-bytes (24-bit)	ECSPI_CS_SEL (SPI only) 00 - CS#0 (default) 01 - CS#1		RECOVER_ECSPI_BOOT_EN '0' - Disabled	DCACHE_BYPASS_DIS

Table continues on the next page...

Table 5-2. Boot Fusemap (continued)

Addr	7	6	5	4	3	2	1	0
				1 - 2-bytes (16-bit)	10 - CS#2 11 - CS#3		'1' - Enabled	
0x490[7:0]	USDHC_PWR_EN 0 - No power cycle 1 - Enabled	EMMC_FAST_BT 0 - Regular 1 - Fast Boot	SDMMC_BUS_WIDTH 00 - 8-bit 01 - 4-bit 10 - 8-bit DDR (MMC 4.4) 11 - 4-bit DDR (MMC 4.4)		SD_SPEED: 00 - Normal/SDR12 01 - High/SDR25 10 - SDR50 11 - SDR104 EMMC_SPEED: 00 - Normal 01 - High		USDHC_VOL_SEL For Normal Boot Mode IO Voltage 0 - 3.3V 1 - 1.8V	USDHC_MFG_VOL_SEL For Mfg Mode IO Voltage 0 - 3.3V 1 - 1.8V
0x490[15:8]	USDHC_DLL_SEL 0 - DLL Slave Mode 1 - DLL Override Mode	SDMMC_DLL_DLY[6:0] Delay target for USDHC DLL, it is applied to slave mode target delay or override mode target delay depends on DLL Override fuse bit value.						
0x490[23:16]	RECOVERY_SDMMC_BOOT_DIS 0 - Enable 1 - Disable	IMG_CNTN_SET1_OFFSET				USDHC_PAD_SION_EN 0 - Disable 1 - Enable	Reserved	USDHC_DLL_EN 0 - Disable DLL for SD/eMMC 1 - Enable DLL for SD/eMMC
0x490[31:24]	USDHC_OVRD_PAD_SETTING_LOW8 [7:0]							
0x4A0[7:0]	SD_CALI_STEP '00' - 1		USDHC_PWR_INTERVAL 00 - 20ms 01 - 10ms 10 - 5ms 11 - 2.5	USDHC_PWR_DELAY 0 - 5ms 1 - 2.5ms	USDHC_PWR_POLARITY 0 - Low 1 - High		USDHC_OVRD_PAD_SETTING_UP1	EMMC_FAST_BT_ACK 0 - Boot Ack Disabled 1 - Boot Ack Enabled
0x4A0[15:8]	BT_TOGGLE_MODE	NAND_FCB_SERCH_COUNT 00 - 2 01 - 2 10 - 4 11 - 8	NAND_TG_PREAMBLE_RD_LATENCY (Toggle Mode 33MHz Preamble Delay, Read Latency) '000' - 16 GPMICLK cycles. '001' - 1 GPMICLK cycles. '010' - 2 GPMICLK cycles. '011' - 3 GPMICLK cycles. '100' - 4 GPMICLK cycles. '101' - 5 GPMICLK cycles.					NAND_RST_TIME

Table continues on the next page...

Table 5-2. Boot Fusemap (continued)

Addr	7	6	5	4	3	2	1	0
				'110' - 6 GPMICLK cycles. '111' - 7 GPMICLK cycles. '1111'- 15 GPMICLK cycles.				
0x4A0[23:16]	Reserved							
0x4A0[31:24]	NAND_OVERRIDE_PAD_SETTING[7:0]							
0x4B0[7:0]	Reserved	NAND_GPMI_DDR_DLL_VAL (GPMI Read DDR DLL Target Value) 0000 - 7 0001 - 1 0111 - 0 1111 - 15				Reserved	NAND_CS_NUM (Nand Number Of Devices) 00 - 1 01 - 2 10 - 4 11 - Reserved	
0x4B0[15:8]	NAND_READ_RETRY_SEQ_ID[3:0] 0000 - Do not use read retry (RR) sequence embedded in ROM 0001 - Micron 20nm RR sequence 0010 - Toshiba A19nm RR sequence 0011 - Toshiba 19nm RR sequence 0100 - SanDisk 19nm RR sequence 0101 - SanDisk 1ynmRR sequence 0110 - Hynix 20nm A Die RR sequence 0111 - Hynix 26nm RR sequence 1000 - Hynix 20nm B Die RR sequence 1001 - Hynix 20nm C Die RR sequence Others - Reserved				NAND_ROW_ADDR_BYTES 00 - 3 01 - 2 10 - 4 11 - 5		Reserved	Reserved
0x4B0[23:16]	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
0x4B0[31:24]	RNG_TRIM[7:0]							

5.2 Lock Fusemap

Table 5-3 describes the functions of various lock fuses.

Table 5-3. Lock Fuses

Addr	7	6	5	4	3	2	1	0
0x400[7:0]	Reserved		Reserved		BOOT_CFG_LOCK 1x - OP x1 - WP		TESTER_LOCK 1x - OP x1 - WP	

Table continues on the next page...

Table 5-3. Lock Fuses (continued)

Addr	7	6	5	4	3	2	1	0
0x400[15:8]	MAC_ADDR_LOCK 1x - OP x1 - WP		USB_ID_LOCK 1x - WP + OP 01 - WP		Reserved	SJC_RESP_LOCK WRP,OP,RDP	SRK_LOCK 1 - WP, OP	Reserved
0x400[23:16]	GP2_LOCK 1x - OP x1 - WP		GP1_LOCK 1x - OP x1 - WP		Reserved		Reserved	Reserved
0x400[31:24]	Reserved						GP5_LOCK 1x – OP x1 – WP	

NOTE

TESTER_LOCK programmed by NXP / set at factory

5.3 Fusemap Descriptions Table

NOTE

Definitions for fuse settings are as follows:

- Unlock - The controlled field can be read, sensed, burned, or overridden in the corresponding OCOTP shadow register.
- Lock - The controlled field cannot be read, burned, or overridden.
- Override Protect (OP) - The controlled field can be read, sensed, or burned in the corresponding OCOTP shadow register.
- Write Protect (WP) - The controlled field can be read, sensed, or overridden in the corresponding OCOTP shadow register.
- OP + WP - The controlled field can only be read or sensed in the corresponding OCOTP shadow register. It cannot be burned or overridden in the corresponding OCOTP shadow register.

Table 5-4. Fusemap Descriptions

Fuse Address	Fuses Name	Number of Fuses	Fuses Function	Setting	Used by
0x400[1:0]	TESTER_LOCK	2	Lock for tester related fuses at 0x410-0x460.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[3:2]	BOOT_CFG_LOCK	2	Lock for BOOT related fuses at 0x470-4B0.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[8:4]	Reserved	5	Reserved	Reserved	Reserved
0x400[9]	SRK_LOCK	1	Lock for SRK_HASH[255:0] fuses.	0 - Unlock 1 - OP + WP	OCOTP
0x400[10]	SJC_RESP_LOCK	1	Lock for SJC_RESP[55:0] fuses.	0 - Unlock 1 - Lock	OCOTP
0x400[11]	Reserved	1	Reserved	Reserved	Reserved
0x400[13:12]	USB_ID_LOCK	2	Lock for USB_PID and USB_VID fuses.	00 - Unlock 01 - WP 11 - OP + WP	OCOTP
0x400[15:14]	MAC_ADDR_LOCK	2	Lock for MAC_ADDR fuses.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[19:16]	Reserved	4	Reserved	Reserved	Reserved
0x400[21:20]	GP1_LOCK	2	Lock for GP1[63:0] fuses.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[23:22]	GP2_LOCK	2	Lock for GP2 fuses.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[25:24]	GP5_LOCK	2	Lock for GP5 fuses.	00 - Unlock 10 - OP 01 - WP 11 - OP + WP	OCOTP
0x400[31:26]	Reserved	6	Reserved	Reserved	Reserved
0x410-0x420	SJC_CHALL[63:0] / UNIQUE_ID[63:0]	32	The SJC CHALLENGE / Unique ID	-	SJC, SW

Table continues on the next page...

Table 5-4. Fusemap Descriptions (continued)

<table>
<tr><th>Fuse Address</th><th>Fuses Name</th><th>Number of Fuses</th><th>Fuses Function</th><th>Setting</th><th>Used by</th></tr>
<tr><td>0x430</td><td>Reserved</td><td>32</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0x440[3:0]</td><td>Reserved</td><td>4</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0x440[7:4]</td><td>Reserved</td><td>4</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0x440[13:8]</td><td>SPEED_GRADING[5:0]</td><td>6</td><td>Burned by tester program, for indicating IC core speed. (Hot burn may not be used).</td><td>
<table>
<tr><th>SPEED_GRADING[5:4]</th><th>SPEED_GRADING[3:0]</th><th>MHz</th><th>P/N Code</th></tr>
<tr><td>xx</td><td>0000</td><td>2300</td><td>23</td></tr>
<tr><td>xx</td><td>0001</td><td>2200</td><td>22</td></tr>
<tr><td>xx</td><td>0010</td><td>2100</td><td>21</td></tr>
<tr><td>xx</td><td>0011</td><td>2000</td><td>20</td></tr>
<tr><td>xx</td><td>0100</td><td>1900</td><td>19</td></tr>
<tr><td>xx</td><td>0101</td><td>1800</td><td>18</td></tr>
<tr><td>xx</td><td>0110</td><td>1700</td><td>17</td></tr>
<tr><td>xx</td><td>0111</td><td>1600</td><td>16</td></tr>
<tr><td>xx</td><td>1000</td><td>1500</td><td>15</td></tr>
<tr><td>xx</td><td>1001</td><td>1400</td><td>14</td></tr>
<tr><td>xx</td><td>1010</td><td>1300</td><td>13</td></tr>
<tr><td>xx</td><td>1011</td><td>1200</td><td>12</td></tr>
<tr><td>xx</td><td>1100</td><td>1100</td><td>11</td></tr>
<tr><td>xx</td><td>1101</td><td>1000</td><td>10</td></tr>
<tr><td>xx</td><td>1110</td><td>900</td><td>09</td></tr>
<tr><td>xx</td><td>1111</td><td>800</td><td>08</td></tr>
</table>
</td><td>PROD / SW</td></tr>
<tr><td>0x440[31:14]</td><td>Reserved</td><td>18</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0x450[1:0]</td><td>NUM_A53_CORES</td><td>2</td><td>Number of A53 CPU cores available.</td><td>00 - 4 cores
01 - Reserved
10 - 2 cores
11 - 1 core</td><td>SRC, SJC, SW</td></tr>
<tr><td>0x450[7:2]</td><td>Reserved</td><td>6</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0x450[8]</td><td>M7_DISABLE</td><td>1</td><td>Disable M7 Core.</td><td>0 - enabled
1 - disabled</td><td>M7</td></tr>
<tr><td>0x450[9]</td><td>M7_MPU_DISABLE</td><td>1</td><td>Disable M7 MPU IP.</td><td>0 - enabled
1 - disabled</td><td>M7</td></tr>
<tr><td>0x450[10]</td><td>M7_FPU_DISABLE</td><td>1</td><td>Disable M7 FPU IP.</td><td>0 - enabled
1 - disabled</td><td>M7</td></tr>
</table>

Table continues on the next page...

Table 5-4. Fusemap Descriptions (continued)

Fuse Address	Fuses Name	Number of Fuses	Fuses Function	Setting	Used by
0x450[11]	USB_OTG1_DISABLE	1	Disable USB OTG1 IP.	0 - enabled 1 - disabled	USB OTG1
0x450[15:12]	Reserved	4	Reserved	Reserved	Reserved
0x450[16]	EXPORT_CONTROL	1	Used for disabling CAAM and SNVS encryption	0 - Secure part 1 - Security disable (CAAM encryption disabled)	CAAM, SW(ROM)
0x450[17]	SEC_CONFIG[0]	1	Security Configuration Mode, and block off debugging of security HW, by JTAG.	Combined with SEC_CONFIG[1], provide FAB/Open/Close security states: 00 - FAB (Open) 01 - Open - allows any code to be flashed and executed, even if it has no valid signature. 1x - Closed (Security On) Also - used for 'blocking' debug of security modules, when burned.	SJC, SNVS, , SRC, TPSMP
0x450[23:18]	Reserved	6	Reserved.	Reserved	Reserved
0x450[24]	GPU3D_DISABLE	1	Disable GPU 3D IP.	0 - enabled 1 - disabled	GPU
0x450[27:25]	Reserved	3	Reserved	Reserved	Reserved
0x450[28]	MIPI_DSI_DISABLE	1	Disable MIPI DSI IP.	0 - enabled 1 - disabled	MIPI DSI
0x450[29]	ENET_DISABLE	1	Disable ENET IP.	0 - enabled 1 - disabled	ENET
0x450[30]	MIPI_CSI_DISABLE	1	Disable MIPI CSI IP.	0 - enabled 1 - disabled	MIPI CSI
0x450[31]	ASRC_DISABLE	1	Disable Audio ASRC IP.	0 - enabled 1 - disabled	ASRC
0x460[31:0]	Reserved	32	Reserved	Reserved	Reserved
0x470[15:0]	BOOT_CFG	16	BOOT configuration register, Usage varies, depending on selected boot device.	See boot fusemap for details.	SRC SW(ROM)
0x470[24:16]	Reserved	9	Reserved	Reserved	Reserved
0x470[19:16]	Reserved	4	Reserved	Reserved	Reserved
0x470[20]	KTE	1	Kill Trace Enable. Enables tracing capability on ETM, and other modules.	0 - Bus tracing is allowed 1 - Bus tracing is allowed in case security state as defined by Secure JTAG	SJC

Table continues on the next page...

Table 5-4. Fusemap Descriptions (continued)

Fuse Address	Fuses Name	Number of Fuses	Fuses Function	Setting	Used by
				allows it (for example, JTAG_ENABLE or NO_DEBUG)	
0x470[21]	SJC_DISABLE	1	Disable/Enable the Secure JTAG Controller module. This fuse is used to create highest JTAG security level, where JTAG is totally blocked.	0 - Secure JTAG Controller is enabled 1 - Secure JTAG Controller is disabled	SJC
0x470[23:22]	JTAG_SMODE	2	JTAG Security Mode. Controls the security mode of the JTAG debug interface	00 - JTAG enable mode 01 - Secure JTAG mode 11 - No debug mode	SJC
0x470[24]	Reserved	1	Reserved	Reserved	Reserved
0x470[25]	SEC_CONFIG[1]	1	Security Configuration (with SEC_CONFIG[0])	00 - FAB (Open) 01 - Open - allows any code to be flashed and executed, even if it has no valid signature. 1x - Closed (Security On)	SW (ROM), SRC, SNVS, TPSMP
0x470[26]	JTAG_HEO	1	JTAG HAB Enable Override. Disallows HAB JTAG enabling. The HAB may normally enable JTAG debugging by means of the OCOTP[HAB_JDE] bit. The JTAG_HEO bit can override this behavior.	0 - HAB may enable JTAG debug access 1 - HAB JTAG enable is overridden (HAB may not enable JTAG debug access)	OCOTP
0x470[27]	Reserved	1	Reserved	Reserved	Reserved
0x470[28]	BT_FUSE_SEL	1	Determines, whether using fuses for boot configuration, or GPIO /Serial loader.	If boot_mode="00" (Development) 0=Boot mode configuration is taken from GPIOs. 1=Boot mode configuration is taken from fuses. If boot_mode="10" (Production) 0 - Boot using Serial Loader (USB) 1- Boot mode configuration is taken from fuses.	SRC SW(ROM)
0x470[29]	FORCE_COLD_BOOT(SBMR)	1	Force cold boot when A7 core come out of reset. Reflected in SBMR reg of SRC	Fuse Function: 0 – Default behavior allowing a fast recovery from low power modes. That is, the ROM is allowed to jump to	SRC SW(ROM)

Table continues on the next page...

Table 5-4. Fusemap Descriptions (continued)

Fuse Address	Fuses Name	Number of Fuses	Fuses Function	Setting	Used by
				the address previously programmed in the SRC persistent register. 1 – Fast recovery path in the ROM is not allowed and a cold boot is always performed. Customers wanting a higher level of security should burn this fuse.	
0x470[31:30]	Reserved	2	Reserved	Reserved	Reserved
0x480[31:0]	BOOT_CFG_PARAMETER	32	BOOT configuration parameters, Usage varies, depending on selected boot device.	See boot fusemap for details.	SW (ROM)
0x490[31:0]	BOOT_CFG_PARAMETER	32	BOOT configuration parameters, Usage varies, depending on selected boot device.	See boot fusemap for details.	SW (ROM)
0x4A0[31:0]	BOOT_CFG_PARAMETER	32	BOOT configuration parameters, Usage varies, depending on selected boot device.	See boot fusemap for details.	SW (ROM)
0x4B0[31:0]	BOOT_CFG_PARAMETER	32	BOOT configuration parameters, Usage varies, depending on selected boot device.	See boot fusemap for details.	SW (ROM)
0x4C0-0x500	Reserved	384	Reserved	Reserved	Reserved
0x580[31:0]	SRK_HASH[255:0]	256	SRK key, no HW visible lines. NO HW Visible signals available	-	SW (HAB)
0x600[23:0]	SJC_RESP[55:0]	56	Response reference value for the secure JTAG controller	-	SJC
0x610[31:24]	Reserved	8	Reserved	Reserved	Reserved
0x620[15:0]	USB_VID[31:0]	16	USB VID	-	SW
0x620[31:16]	USB_PID[31:0]	16	USB PID	-	SW
0x630[0]	FIELD_RETURN	1	Configure device for field return testing. Fuse burning is protected by CSF command, with proper parameter passed. Non-lockable	0 - Device is in functional / secure mode. 1 - Device is open for 'field-return' testing.	SW (ROM), SNVS_HP, SRC, TPSMP, Security Logic
0x630[31:1]	Reserved	31	Reserved	Reserved	Reserved
0x640[15:0]	MAC_ADDR[47:0]	48	Reserved for customers/SW	-	SW
0x650[31:16]	Reserved	48	Reserved	Reserved	Reserved

Table continues on the next page...

Table 5-4. Fusemap Descriptions (continued)

Fuse Address	Fuses Name	Number of Fuses	Fuses Function	Setting	Used by
0x670[3:0]	SRK_REVOKE	4	SRK keys recokation	Full support to revoke up to 4 SRK keys. 0000 - No Revoke 0001 - Key 1 0010 - Key 2 0011 - Key 1 and 2 0100 - Key 3 0101 - Key 1 and 2 0110 - Key 2 and 3 0111 - Key 1, 2, and 3 1000 - Key 4 1001 - Key 1 and 4 ... 1111 - All Keys	SW(HAB)
0x670[7:4]	Reserved	4	Reserved	Reserved	Reserved
0x670[15:8]	MC_ERA	8	SNVS Monolitic Counter Era value	See SNVS spec	SNVS
0x670[31:16]	AP_BI_VER	16	Application Processor Boot Image Version. Indicate the version of the code image authenticated by the High Assurance Boot. The fuse value must match the version of the image stored in non-volatile memory.	-	SW
0x670-0x6F0	Reserved	256	Reserved	Reserved	Reserved
0x700[31:0]	Reserved	256	Reserved	Reserved	Reserved
0x780[31:0]	GP1[63:0]	64	General Purpose fuse register #1	-	SW
0x7A0[31:0]	GP2[63:0]	64	General Purpose fuse register #2	-	SW
0x7C0-0x7F0	Reserved	128	Reserved	Reserved	Reserved
0x800[31:0]	GP5[383:0]	384	General Purpose fuse register #5	-	SW
0x8C0[31:0]	Reserved	128	Reserved	Reserved	Reserved
0x900-0x13F0	Reserved	5632	Reserved	Reserved	Reserved

Chapter 6
On-Chip OTP Controller (OCOTP_CTRL)

6.1 Overview

This section contains information describing the requirements for the on-chip eFuse OTP controller along with details about the block functionality and implementation.

In this document, the words "eFuse" and "OTP" are interchangeable. OCOTP refers to the hardware block itself.

6.1.1 Features

The OCOTP provides the following features:

- Loading and housing of fuse content into shadow registers.
- Generation of HWV_FUSE (hardware visible fuse bus) and the HWV_REG bus which is made up of volatile PIO register based "fuses". The HWV_REG bits come from the SCS (Software Controllable Signals) register.
- Generation of STICKY_REG which is consist of sticky register bits.
- Provide program-protect and read-protect eFuse.
- Provide override and read protection of shadow register.

6.2 Top-Level Symbol and Functional Overview

The figure found here shows the OCOTP system level diagram.

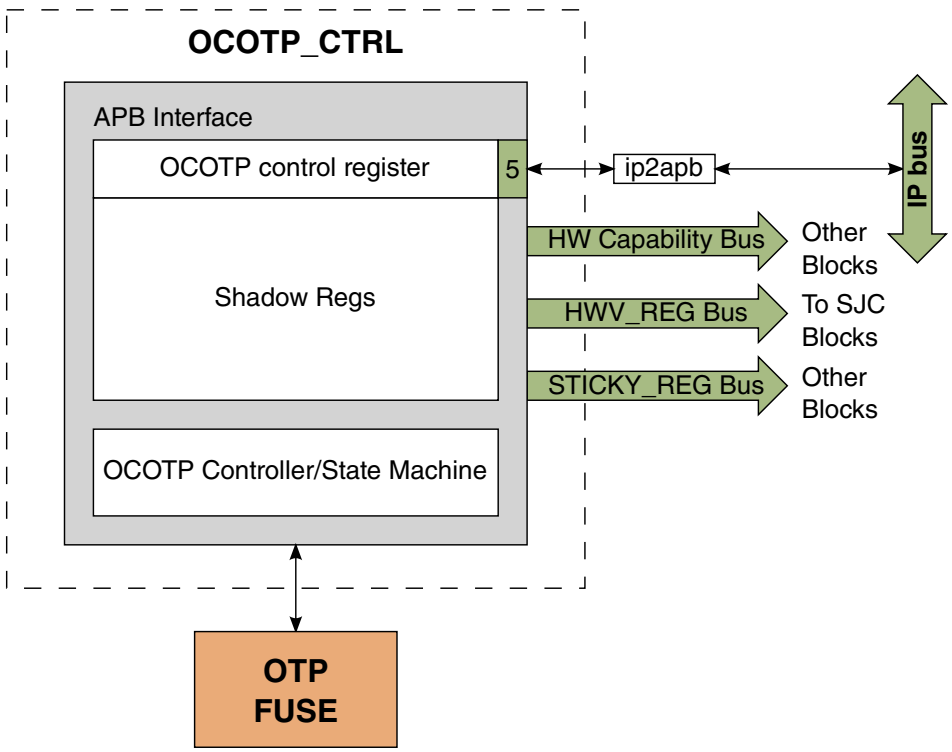


Figure 6-1. OCOTP System Level Diagram

6.2.1 Operation

The IP bus interface of the OCOTP provides two functions.

- Configure control registers for programming and reading fuse .
- Override and read shadow registers.

6.2.1.1 Shadow Register Reload

All fuse words in are shadowed. Therefore, fuse information is available through memory mapped shadow registers. If fuses are subsequently programmed, the shadow registers should be reloaded to keep them coherent with the fuse bank arrays.

The "reload shadows" feature allows the user to force a reload of the shadow registers (including HW_OCOTP_LOCK) without having to reset the device. To force a reload, complete the following steps:

1. Check that HW_OCOTP_CTRL[BUSY] and HW_OCOTP_CTRL[ERROR] are clear. Overlapped accesses are not supported by the controller. Any pending write , read or reload must be completed before a new access can be requested.
2. Set the HW_OCOTP_CTRL[RELOAD_SHADOWS] bit. OCOTP will read all the fuse one by one and put it into corresponding shadow register.
3. Wait for HW_OCOTP_CTRL[BUSY] and HW_OCOTP_CTRL[RELOAD_SHADOWS] to be cleared by the controller.

The controller will automatically clear the HW_OCOTP_CTRL[RELOAD_SHADOWS] bit after the successful completion of the operation.

6.2.1.2 Fuse and Shadow Register Read

All shadow registers are always readable through the APB bus except some secret keys regions. When their corresponding fuse lock bits are set, the shadow registers also become read locked. After read locking, reading from these registers will return 0xBADABADA.

In addition HW_OCOTP_CTRL[ERROR] will be set. It must be cleared by software before any new write , read or reload access can be issued. Subsequent reads to unlocked shadow locations will still work successfully however.

To read fuse words directly from correctly complete the following steps:

1. Check that HW_OCOTP_CTRL[BUSY] and HW_OCOTP_CTRL[ERROR] are clear. Overlapped accesses are not supported by the controller. Any pending write, read or reload must be completed before a read access can be requested.
2. Write the requested to HW_OCOTP_CTRL[ADDR].

6.2.1.3 Fuse and Shadow Register Writes

Shadow register bits can be overridden by software until the corresponding fuse lock bit for the region is set. When the lock shadow bit is set, the shadow registers for that lock region become write locked. The LOCK shadow register also has no shadow or fuse lock bits but it is always read only.

In order to avoid "rogue" code performing erroneous writes to OTP, a special unlocking sequence is required for writes to the fuse banks. To program fuse bank correctly complete the following steps:

1. Program HW_OCOTP_TIMING[STROBE_PROG] and fields with timing values to match the current frequency of the ipg_clk. OTP writes will work at maximum bus frequencies as long as the parameters are set correctly.
2. Check that HW_OCOTP_CTRL[BUSY] and HW_OCOTP_CTRL[ERROR] are clear. Overlapped accesses are not supported by the controller. Any pending write or reload must be completed before a write access can be requested.
3. Write the requested to HW_OCOTP_CTRL[ADDR] and program the unlock code into HW_OCOTP_CTRL[WR_UNLOCK]. This must be programmed for each write access. The lock code is documented in the register description. Both the unlock code and address can be written in the same operation.

It should be noted that write latencies to OTP are numbers of 10 micro-seconds. Write latencies is based on amount of bit filed which is 1. For example : program half fuse bits in one word need 10 us x 16.

For further details of OTP read/write operations see [eFUSE].

HW_OCOTP_CTRL[ERROR] will be set under the following conditions:

- A write is performed to a shadow register during a shadow reload (essentially, while HW_OCOTP_CTRL[RELOAD_SHADOWS] is set. In addition, the contents of the shadow register shall not be updated.
- A write is performed to a shadow register which has been locked.
- A read is performed to from a shadow register which has been read locked.
- A program is performed to a fuse which has been .
- A read is performed to from a fuse which has been read locked.

6.2.1.4 Write Postamble

Due to internal electrical characteristics of the OTP during writes, all OTP operations following a write must be separated by 2 us after the clearing of HW_OCOTP_CTRL_BUSY following the write. This guarantees programming voltages on-chip to reach a steady state when exiting a write sequence. This includes reads, shadow reloads, or other writes.

A recommended software sequence to meet the postamble requirements is as follows:

- Issue the write and poll for BUSY (as per Fuse Shadow Memory Footprint).
- After BUSY is clear, wait an additional 2 us.
- Perform the next OTP operation.

6.2.2 Fuse Shadow Memory Footprint

The OTP memory footprint shows in the following figure. The registers are grouped by lock region. Their names correspond to the PIO register and fusemap names.

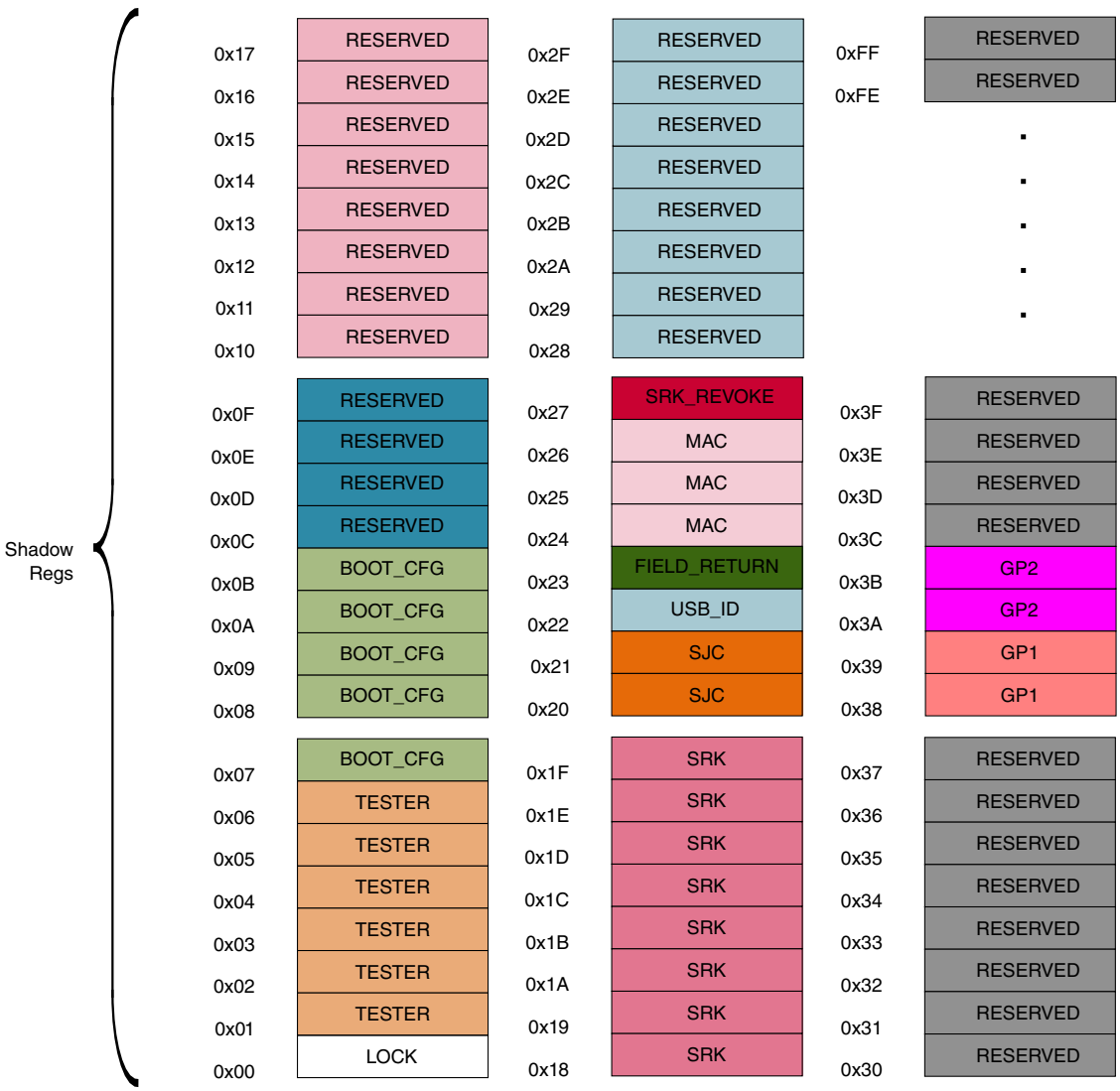


Figure 6-2. OTP Memory Footprint

6.2.3 OTP Read/Write Timing Parameters

There are timing fields contained in the HW_OCOTP_TIMING register that specify counter limit values, which are used to specify the signal timing.

Both two timing parameters are specified in ipg_clk cycles. Since the ipg_clk frequency can be set to a range of values, these parameters must be adjusted with the clock to yield the appropriate delay.

6.2.4 Hardware Visible Fuses

The hwv_fuse bus emanates from the OCOTP block and goes to various other blocks inside the chip. This bus is made up of the shadow register bits for .

Only a subset of these fuse bits are currently used by the hardware. The fuse bits are initially copied from the banks after reset is deasserted. When all fuse bits are loaded into their shadow registers, the OCOTP asserts the fuse_latched output signal.

The hwv_reg bus also comes from the OCOTP. Its source is the HW_OCOTP_SCS register. This register has 1 defined bit, the HAB_JDE bit, that is connected to the SJC block. The SCS bits are intended to be used as volatile fuse bits under software control. Additional bits will be defined as needed in future implementations.

The system-wide reset sequence must be coordinated by the system reset controller, so that the hwv_fuse and hwv_reg buses are stable and reflect the values of the fuses before they are used by the rest of the system.

6.2.5 Behavior During Reset

The OCOTP is always active. The shadow registers automatically load the appropriate OTP contents after reset is deasserted. During this load-time HW_OCOTP_CTRL_BUSY is set. The load time is similar to that of a "reload shadow" operation.

6.2.6 Secure JTAG control

The JTAG control fuses are used to allow or disallow JTAG access to secured resources.

Three JTAG security levels are envisioned, as shown in the table below.

Table 6-1. JTAG Security Level Control Bits

Security Mode	JTAG_SMODE	Description
No Debug	2'b11	The highest security level.
Secure JTAG	2'b01	Limit the JTAG access by using key based authentication mechanism.
JTAG Enable	2'b00	Low Security, all JTAG features are enabled.

6.3 Fuse Map

See the Fusemap chapter of this reference manual for more information.

6.4 OCOTP Memory Map/Register Definition

NOTE

When write/read unimplemented register address in ocotp_ctrl, ocotp_ctrl will not send error and read data will be 0.

OCOTP Hardware Register Format Summary

OCOTP memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
0	OTP Controller Control Register (OCOTP_HW_OCOTP_CTRL)	32	R/W	0000_0000h	6.4.1/174
4	OTP Controller Control Register (OCOTP_HW_OCOTP_CTRL_SET)	32	R/W	0000_0000h	6.4.1/174
8	OTP Controller Control Register (OCOTP_HW_OCOTP_CTRL_CLR)	32	R/W	0000_0000h	6.4.1/174
C	OTP Controller Control Register (OCOTP_HW_OCOTP_CTRL_TOG)	32	R/W	0000_0000h	6.4.1/174
10	OTP Controller Timing Register (OCOTP_HW_OCOTP_TIMING)	32	R/W	0400_0000h	6.4.2/175
20	OTP Controller Write Data Register (OCOTP_HW_OCOTP_DATA)	32	R/W	0000_0000h	6.4.3/176
30	OTP Controller Write Data Register (OCOTP_HW_OCOTP_READ_CTRL)	32	R/W	0000_0000h	6.4.4/177
40	OTP Controller Read Data Register (OCOTP_HW_OCOTP_READ_FUSE_DATA)	32	R/W	0000_0000h	6.4.5/178
50	Sticky bit Register (OCOTP_HW_OCOTP_SW_STICKY)	32	R/W	0000_0000h	6.4.6/179
60	Software Controllable Signals Register (OCOTP_HW_OCOTP_SCS)	32	R/W	0000_0000h	6.4.7/180
64	Software Controllable Signals Register (OCOTP_HW_OCOTP_SCS_SET)	32	R/W	0000_0000h	6.4.7/180
68	Software Controllable Signals Register (OCOTP_HW_OCOTP_SCS_CLR)	32	R/W	0000_0000h	6.4.7/180
6C	Software Controllable Signals Register (OCOTP_HW_OCOTP_SCS_TOG)	32	R/W	0000_0000h	6.4.7/180

Table continues on the next page...

OCOTP memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
90	OTP Controller Version Register (OCOTP_HW_OCOTP_VERSION)	32	R/W	0148_1299h	6.4.8/181
400	Value of OTP Bank0 Word0 (Lock controls) (OCOTP_HW_OCOTP_LOCK)	32	R/W	0000_0000h	6.4.9/182
410	Value of OTP Bank0 Word1 (Tester Info.) (OCOTP_HW_OCOTP_TESTER0)	32	R/W	0000_0000h	6.4.10/184
420	Value of OTP Bank0 Word2 (tester Info.) (OCOTP_HW_OCOTP_TESTER1)	32	R/W	0000_0000h	6.4.11/184
430	Value of OTP Bank0 Word3 (Tester Info.) (OCOTP_HW_OCOTP_TESTER2)	32	R/W	0000_0000h	6.4.12/185
440	Value of OTP Bank1 Word0 (Tester Info.) (OCOTP_HW_OCOTP_TESTER3)	32	R/W	0000_0000h	6.4.13/185
450	Value of OTP Bank1 Word1 (Tester Info.) (OCOTP_HW_OCOTP_TESTER4)	32	R/W	0000_0000h	6.4.14/186
460	Value of OTP Bank1 Word2 (Tester Info.) (OCOTP_HW_OCOTP_TESTER5)	32	R/W	0000_0000h	6.4.15/186
470	Value of OTP Bank1 Word3 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG0)	32	R/W	0000_0000h	6.4.16/187
480	Value of OTP Bank2 Word0 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG1)	32	R/W	0000_0000h	6.4.17/187
490	Value of OTP Bank2 Word1 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG2)	32	R/W	0000_0000h	6.4.18/188
4A0	Value of OTP Bank2 Word2 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG3)	32	R/W	0000_0000h	6.4.19/188
4B0	Value of OTP Bank2 Word3 (BOOT Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG4)	32	R/W	0000_0000h	6.4.20/189
580	Shadow Register for OTP Bank6 Word0 (SRK Hash) (OCOTP_HW_OCOTP_SRK0)	32	R/W	0000_0000h	6.4.21/189
590	Shadow Register for OTP Bank6 Word1 (SRK Hash) (OCOTP_HW_OCOTP_SRK1)	32	R/W	0000_0000h	6.4.22/190
5A0	Shadow Register for OTP Bank6 Word2 (SRK Hash) (OCOTP_HW_OCOTP_SRK2)	32	R/W	0000_0000h	6.4.23/190
5B0	Shadow Register for OTP Bank6 Word3 (SRK Hash) (OCOTP_HW_OCOTP_SRK3)	32	R/W	0000_0000h	6.4.24/191
5C0	Shadow Register for OTP Bank7 Word0 (SRK Hash) (OCOTP_HW_OCOTP_SRK4)	32	R/W	0000_0000h	6.4.25/191
5D0	Shadow Register for OTP Bank7 Word1 (SRK Hash) (OCOTP_HW_OCOTP_SRK5)	32	R/W	0000_0000h	6.4.26/192
5E0	Shadow Register for OTP Bank7 Word2 (SRK Hash) (OCOTP_HW_OCOTP_SRK6)	32	R/W	0000_0000h	6.4.27/192
5F0	Shadow Register for OTP Bank7 Word3 (SRK Hash) (OCOTP_HW_OCOTP_SRK7)	32	R/W	0000_0000h	6.4.28/193
600	Value of OTP Bank8 Word0 (Secure JTAG Response Field) (OCOTP_HW_OCOTP_SJC_RESP0)	32	R/W	0000_0000h	6.4.29/193

Table continues on the next page...

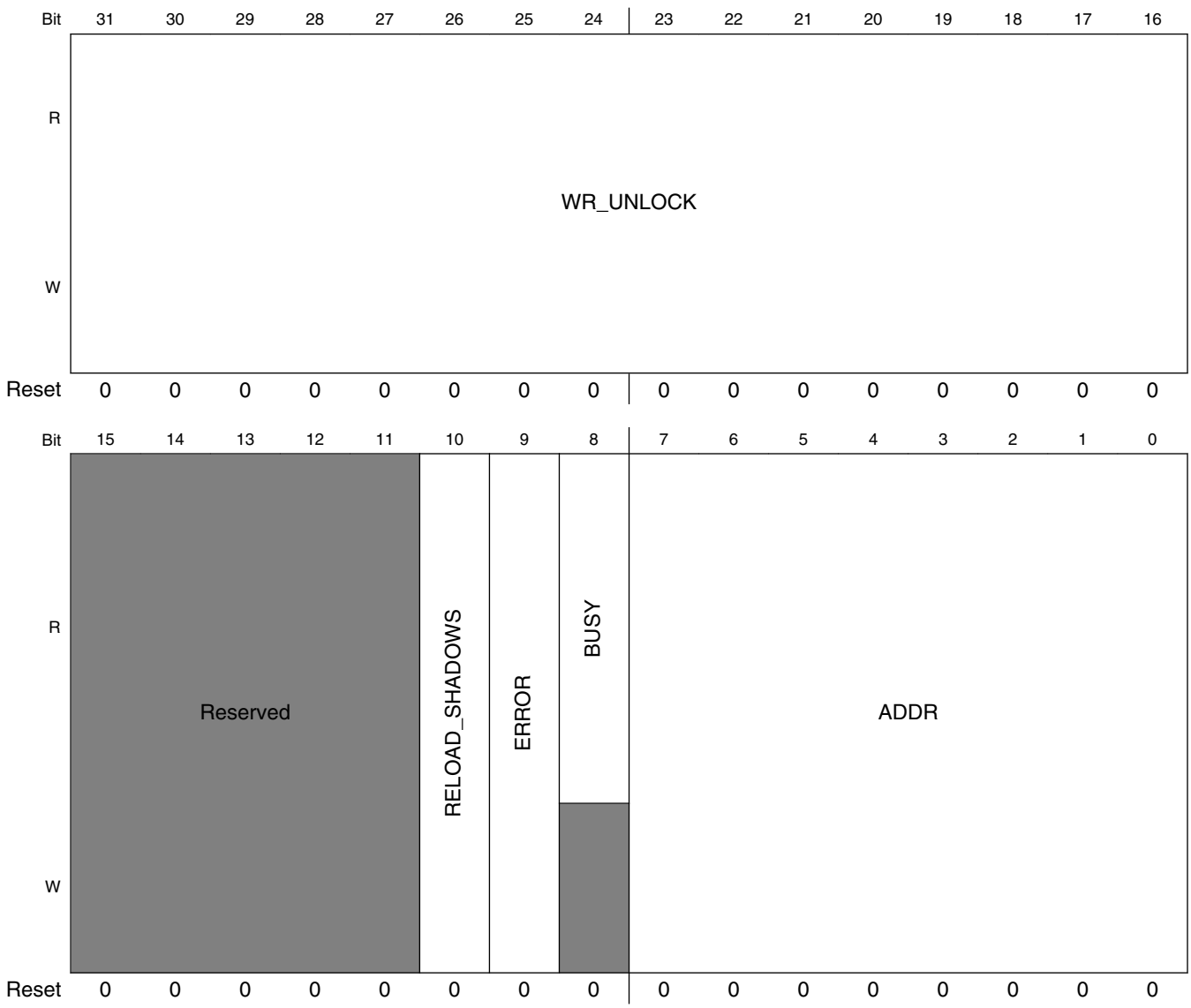
OCOTP memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
610	Value of OTP Bank8 Word1 (Secure JTAG Response Field) (OCOTP_HW_OCOTP_SJC_RESP1)	32	R/W	0000_0000h	6.4.30/194
620	Value of OTP Bank8 Word2 (USB ID info) (OCOTP_HW_OCOTP_USB_ID)	32	R/W	0000_0000h	6.4.31/194
630	Value of OTP Bank8 Word3 (Field Return) (OCOTP_HW_OCOTP_FIELD_RETURN)	32	R/W	0000_0000h	6.4.32/195
640	Value of OTP Bank9 Word0 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR0)	32	R/W	0000_0000h	6.4.33/195
650	Value of OTP Bank9 Word1 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR1)	32	R/W	0000_0000h	6.4.34/196
660	Value of OTP Bank9 Word2 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR2)	32	R/W	0000_0000h	6.4.35/196
780	Value of OTP Bank14 Word0 () (OCOTP_HW_OCOTP_GP10)	32	R/W	0000_0000h	6.4.36/197
790	Value of OTP Bank14 Word1 () (OCOTP_HW_OCOTP_GP11)	32	R/W	0000_0000h	6.4.37/197
7A0	Value of OTP Bank14 Word2 () (OCOTP_HW_OCOTP_GP20)	32	R/W	0000_0000h	6.4.38/197
7B0	Value of OTP Bank14 Word3 () (OCOTP_HW_OCOTP_GP21)	32	R/W	0000_0000h	6.4.39/198

6.4.1 OTP Controller Control Register (OCOTP_HW_OCOTP_CTRL*n*)

The OCOTP Control and Status Register provides the necessary software interface for performing read and write operations to the On-Chip OTP (One-Time Programmable ROM). The control fields such as WR_UNLOCK, ADDR and BUSY/ERROR may be used in conjuction with the HW_OCOTP_DATA register to perform write operations. Read operations to the On-Chip OTP are involving ADDR, BUSY/ERROR bit field and HW_OCOTP_READ_CTRL register. Read value is saved in HW_OCOTP_READ_FUSE_DATA register.

Address: 0h base + 0h offset + (4d × i), where i=0d to 3d



OCOTP_HW_OCOTP_CTRL*n* field descriptions

Field	Description
31–16 WR_UNLOCK	Write 0x3E77 to enable OTP write accesses. NOTE: This register must be unlocked on a write-by-write basis (a write is initiated when HW_OCOTP_DATA is written), so the UNLOCK bitfield must contain the correct key value during all writes to HW_OCOTP_DATA, otherwise a write shall not be initiated. This field is automatically cleared after a successful write completion (clearing of BUSY).
15–11 -	This field is reserved. Reserved
10 RELOAD_ SHADOWS	Set to force re-loading the shadow registers (HW/SW capability and LOCK). This operation will automatically set BUSY. Once the shadow registers have been re-loaded, BUSY and RELOAD_SHADOWS are automatically cleared by the controller.
9 ERROR	Set by the controller when an access to a locked region(OTP or shadow register) is requested. Must be cleared before any further access can be performed. This bit can only be set by the controller. This bit is also set if the Pin interface is active and software requests an access to the OTP. In this instance, the ERROR bit cannot be cleared until the Pin interface access has completed. Reset this bit by writing a one to the SCT clear address space and not by a general write.
8 BUSY	OTP controller status bit. When active, no new write access or read access to OTP(including RELOAD_SHADOWS) can be performed. Cleared by controller when access complete. After reset (or after setting RELOAD_SHADOWS), this bit is set by the controller until the HW/SW and LOCK registers are successfully copied, after which time it is automatically cleared by the controller.
ADDR	OTP write and read access address register. Specifies one of 128 word address locations (0x00 - 0x7f). If a valid access is accepted by the controller, the controller makes an internal copy of this value. This internal copy will not update until the access is complete.

6.4.2 OTP Controller Timing Register (OCOTP_HW_OCOTP_TIMING)

This register specifies timing parameters for programming and reading the OCOTP fuse array.

Address: 0h base + 10h offset = 10h

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	RSRVD0				WAIT						STROBE_READ						RELAX				STROBE_PROG											
W																																
Reset	0	0	0	0	0	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

OCOTP_HW_OCOTP_TIMING field descriptions

Field	Description
31–28 RSRVD0	These bits always read back zero.
27–22 WAIT	This count value specifies time interval between auto read and write access in one time program. It is given in number of ipg_clk periods.
21–16 STROBE_READ	This count value specifies the strobe period in one time read OTP. Trd = ((STROBE_READ+1)- 2*(RELAX +1)) /ipg_clk_freq. It is given in number of ipg_clk periods.

Table continues on the next page...

OCOTP_HW_OCOTP_TIMING field descriptions (continued)

Field	Description
15–12 RELAX	This count value specifies the time to add to all default timing parameters other than the Tpgm and Trd. It is given in number of ipg_clk periods.
STROBE_PROG	This count value specifies the strobe period in one time write OTP. Tpgm = ((STROBE_PROG+1)-2*(RELAX+1)) /ipg_clk_freq. It is given in number of ipg_clk periods.

6.4.3 OTP Controller Write Data Register (OCOTP_HW_OCOTP_DATA)

This register is used in conjuction with HW_OCOTP_CTRL to perform one-time writes to the OTP.

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	DATA																															
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

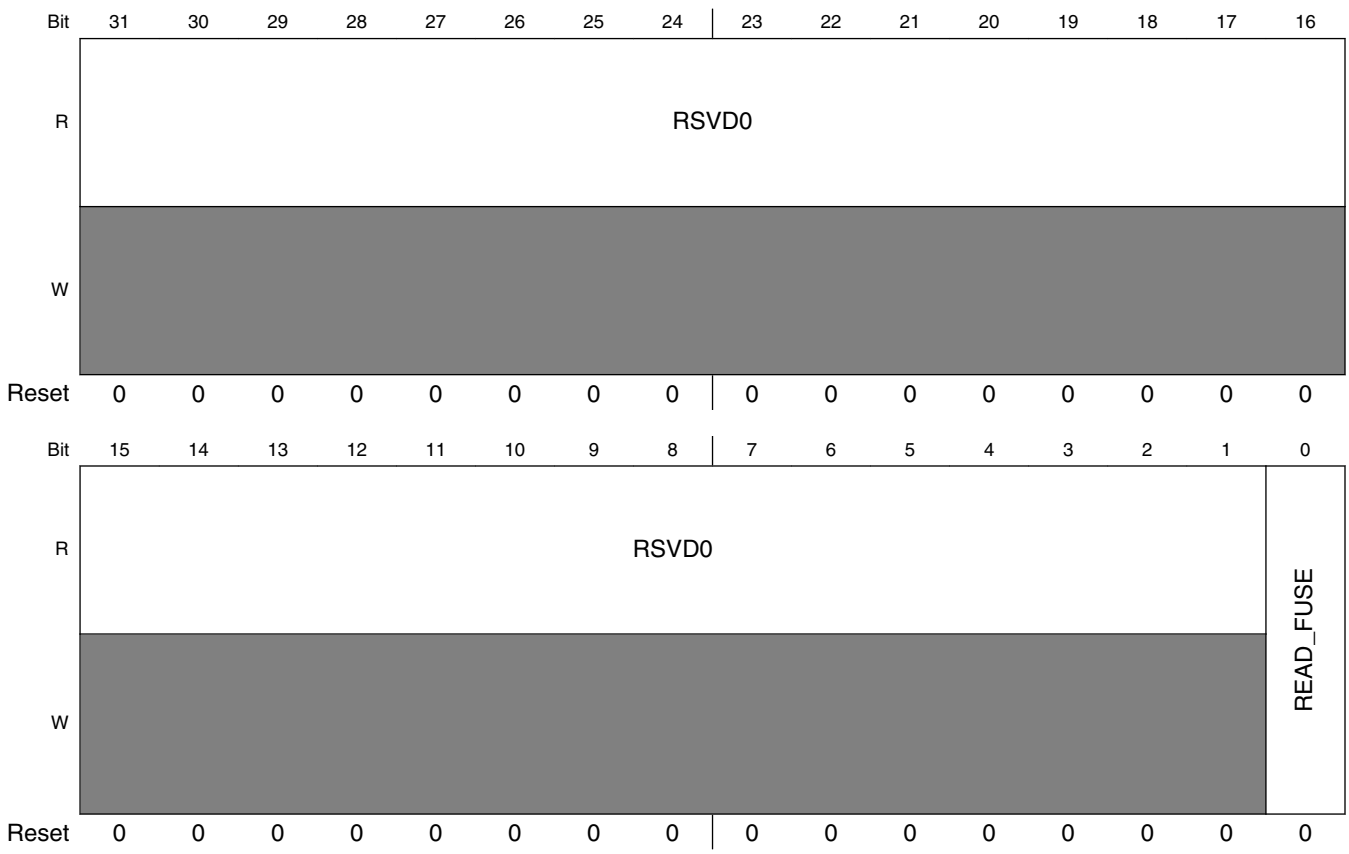
OCOTP_HW_OCOTP_DATA field descriptions

Field	Description
DATA	Used to initiate a write to OTP.

6.4.4 OTP Controller Write Data Register (OCOTP_HW_OCOTP_READ_CTRL)

This register is used in conjuction with HW_OCOTP_CTRL to perform one time read to the OTP.

Address: 0h base + 30h offset = 30h



OCOTP_HW_OCOTP_READ_CTRL field descriptions

Field	Description
31–1 RSVD0	Reserved
0 READ_FUSE	Used to initiate a read to OTP.

6.4.5 OTP Controller Read Data Register (OCOTP_HW_OCOTP_READ_FUSE_DATA)

The data read from OTP

Address: 0h base + 40h offset = 40h

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	DATA																															
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

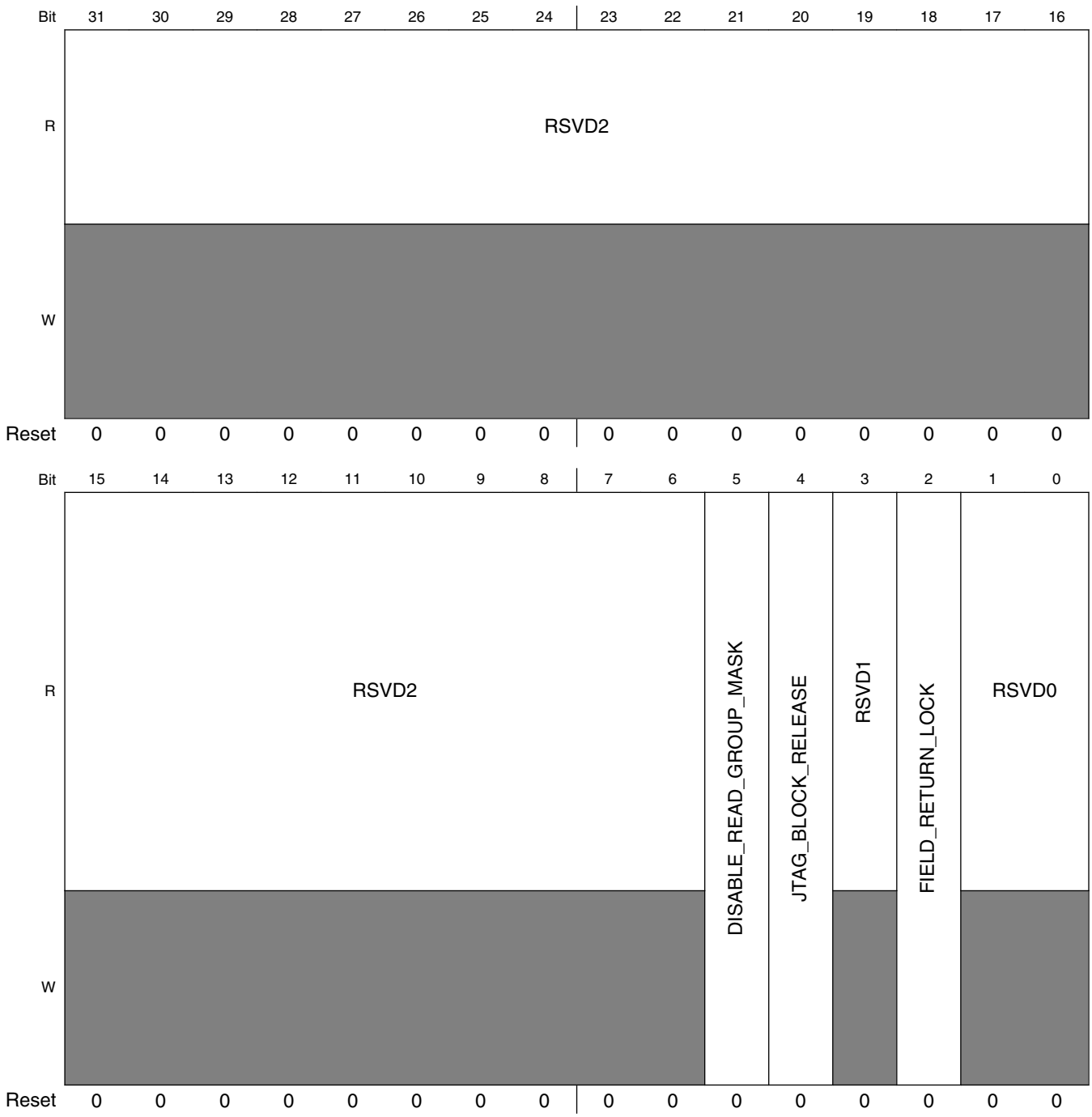
OCOTP_HW_OCOTP_READ_FUSE_DATA field descriptions

Field	Description
DATA	The data read from OTP

6.4.6 Sticky bit Register (OCOTP_HW_OCOTP_SW_STICKY)

Some sticky bits are used by SW to lock some fuse area , shadow registers and other features.

Address: 0h base + 50h offset = 50h



OCOTP_HW_OCOTP_SW_STICKY field descriptions

Field	Description
31–6 RSVD2	Reserved
5 DISABLE_ READ_GROUP_ MASK	Shadow register write and OTP write lock for GROUP_MASK region. When set, the writing of this region's shadow register and OTP fuse word are blocked. Once this bit is set, it is always high unless a POR is issued.
4 JTAG_BLOCK_ RELEASE	Set by Arm during Boot after DTCP is initialized and before test mode entry. * 0 (Default) - JTAG is blocked (subject to other conditions).* 1 - JTAG block is released (subject to other controls). Once this bit is set, it is always high unless a POR is issued.
3 RSVD1	Reserved
2 FIELD_ RETURN_LOCK	Shadow register write and OTP write lock for FIELD_RETURN region. When set, the writing of this region's shadow register and OTP fuse word are blocked.Once this bit is set, it is always high unless a POR is issued.
RSVD0	Reserved

6.4.7 Software Controllable Signals Register (OCOTP_HW_OCOTP_SCS*n*)

This register holds volatile configuration values that can be set and locked by trusted software. All values are returned to their defualt values after POR.

Address: 0h base + 60h offset + (4d × i), where i=0d to 3d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LOCK	SPARE														
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	SPARE															HAB_JDE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

OCOTP_HW_OCOTP_SCS*n* field descriptions

Field	Description
31 LOCK	When set, all of the bits in this register are locked and can not be changed through SW programming. This bit is only reset after a POR is issued.

Table continues on the next page...

OCOTP_HW_OCOTP_SCS*n* field descriptions (continued)

Field	Description
30–1 SPARE	Unallocated read/write bits for implementation specific software use.
0 HAB_JDE	HAB JTAG Debug Enable. This bit is used by the HAB to enable JTAG debugging, assuming that a properlay signed command to do so is found and validated by the HAB. The HAB must lock the register before passing control to the OS whether or not JTAG debugging has been enabled. Once JTAG is enabled by this bit, it can not be disabled unless the system is reset by POR. 0: JTAG debugging is not enabled by the HAB (it may still be enabled by other mechanisms). 1: JTAG debugging is enabled by the HAB (though this signal may be gated off).

6.4.8 OTP Controller Version Register (OCOTP_HW_OCOTP_VERSION)

This register indicates the RTL version in use.

Address: 0h base + 90h offset = 90h

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	MAJOR								MINOR								STEP															
W																																
Reset	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	1

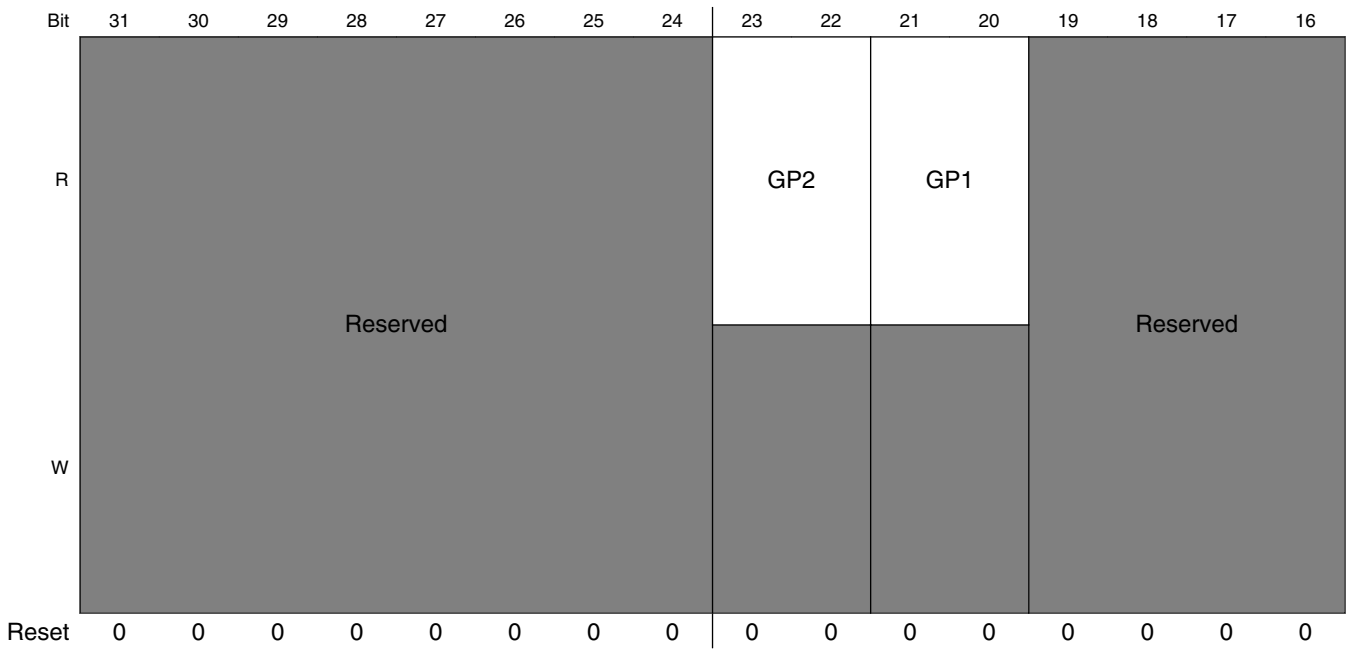
OCOTP_HW_OCOTP_VERSION field descriptions

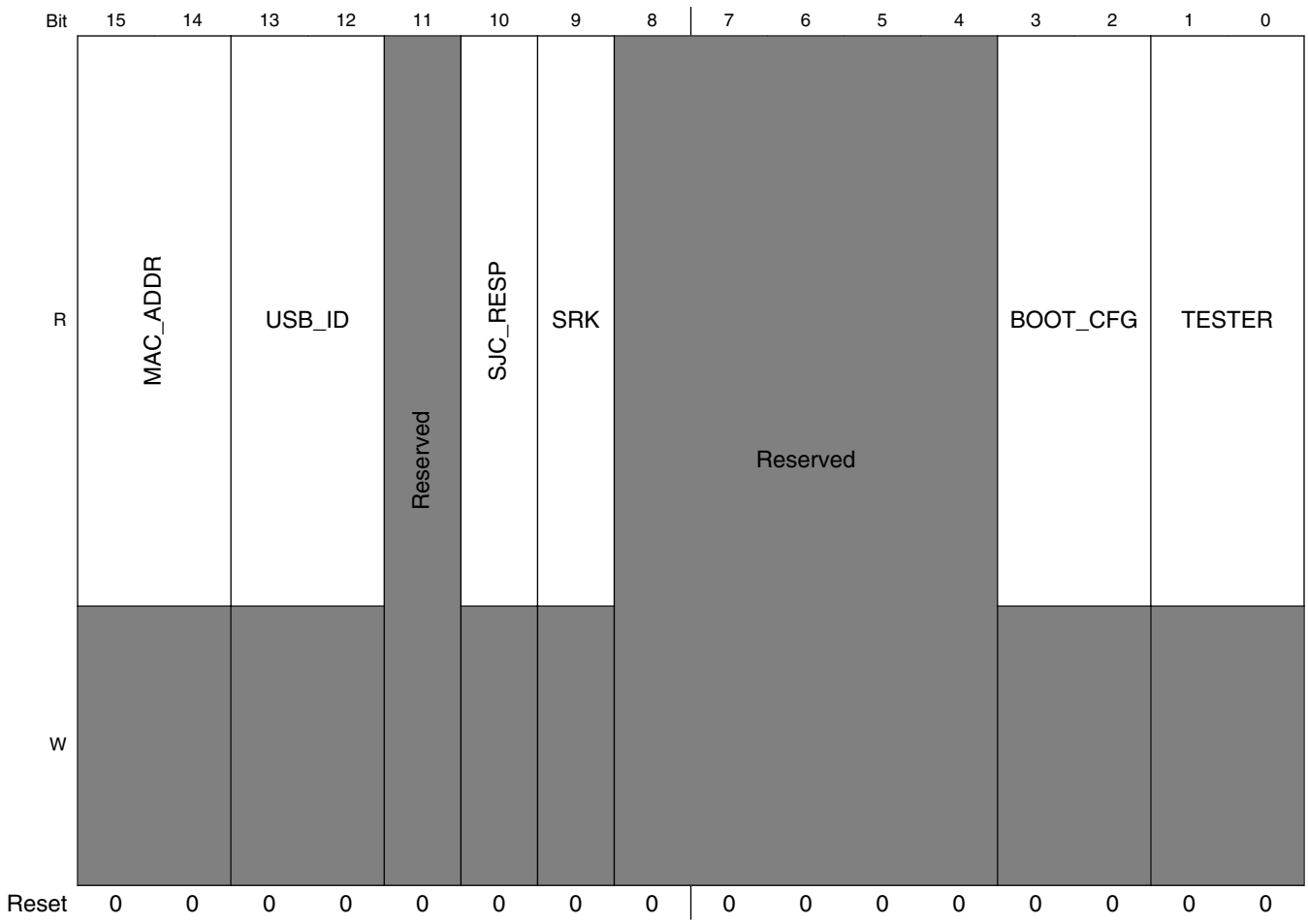
Field	Description
31–24 MAJOR	Fixed read-only value reflecting the MAJOR field of the RTL version.
23–16 MINOR	Fixed read-only value reflecting the MINOR field of the RTL version.
STEP	Fixed read-only value reflecting the stepping of the RTL version.

6.4.9 Value of OTP Bank0 Word0 (Lock controls) (OCOTP_HW_OCOTP_LOCK)

Shadowed memory mapped access to OTP Bank 0, word 0.

Address: 0h base + 400h offset = 400h





OCOTP_HW_OCOTP_LOCK field descriptions

Field	Description
31–24 -	This field is reserved. Reserved
23–22 GP2	Status of shadow register and OTP write lock for gp2 region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.
21–20 GP1	Status of shadow register and OTP write lock for gp1 region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.
19–16 -	This field is reserved. Reserved
15–14 MAC_ADDR	Status of shadow register and OTP write lock for mac_addr region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.
13–12 USB_ID	Status of shadow register and OTP write lock for usb_id region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.
11 -	This field is reserved. Reserved

Table continues on the next page...

OCOTP_HW_OCOTP_LOCK field descriptions (continued)

Field	Description
10 SJC_RESP	Status of shadow register read and write, OTP read and write lock for sjc_resp region. When set, the writing of this region's shadow register and OTP fuse word are blocked. The read of this region's shadow register and OTP fuse word are also blocked.
9 SRK	Status of shadow register and OTP write lock for srk region. When set, the writing of this region's shadow register and OTP fuse word are blocked.
8–4 -	This field is reserved. Reserved
3–2 BOOT_CFG	Status of shadow register and OTP write lock for boot_cfg region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.
TESTER	Status of shadow register and OTP write lock for tester region. When bit 1 is set, the writing of this region's shadow register is blocked. When bit 0 is set, the writing of this region's OTP fuse word is blocked.

6.4.10 Value of OTP Bank0 Word1 (Tester Info.) (OCOTP_HW_OCOTP_TESTER0)

Shadowed memory mapped access to OTP Bank 0, word 1.

Address: 0h base + 410h offset = 410h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER0 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 0, word 1. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.11 Value of OTP Bank0 Word2 (tester Info.) (OCOTP_HW_OCOTP_TESTER1)

shadowed memory mapped access to OTP Bank 0, word 2.

Address: 0h base + 420h offset = 420h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER1 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 0, word 2. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.12 Value of OTP Bank0 Word3 (Tester Info.) (OCOTP_HW_OCOTP_TESTER2)

Shadowed memory mapped access to OTP Bank 0, word 3.

Address: 0h base + 430h offset = 430h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER2 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 0, word 3. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.13 Value of OTP Bank1 Word0 (Tester Info.) (OCOTP_HW_OCOTP_TESTER3)

Non-shadowed memory mapped access to OTP Bank 1, word 0.

Address: 0h base + 440h offset = 440h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER3 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 1, word 0. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.14 Value of OTP Bank1 Word1 (Tester Info.) (OCOTP_HW_OCOTP_TESTER4)

Shadowed memory mapped access to OTP Bank 1, word 1.

Address: 0h base + 450h offset = 450h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER4 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 1, word 1. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.15 Value of OTP Bank1 Word2 (Tester Info.) (OCOTP_HW_OCOTP_TESTER5)

Shadowed memory mapped access to OTP Bank 1, word 2.

Address: 0h base + 460h offset = 460h

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	BITS																															
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

OCOTP_HW_OCOTP_TESTER5 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 1, word 2. These bits become read-only after the HW_OCOTP_LOCK_TESTER[1] bit is set.

6.4.16 Value of OTP Bank1 Word3 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG0)

Shadowed memory mapped access to OTP Bank 1, word 3.

Address: 0h base + 470h offset = 470h

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	BITS																															
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

OCOTP_HW_OCOTP_BOOT_CFG0 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 1, word 3. These bits become read-only after the HW_OCOTP_LOCK_BOOT_CFG[1] bit is set.

6.4.17 Value of OTP Bank2 Word0 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG1)

Shadowed memory mapped access to OTP bank 2, word 0.

Address: 0h base + 480h offset = 480h

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	BITS																															
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

OCOTP_HW_OCOTP_BOOT_CFG1 field descriptions

Field	Description
BITS	Reflects value of OTP bank 2, word 0. These bits become read-only after the HW_OCOTP_LOCK_BOOT_CFG[1] bit is set.

6.4.18 Value of OTP Bank2 Word1 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG2)

Shadowed memory mapped access to OTP bank 2, word 1.

Address: 0h base + 490h offset = 490h

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	BITS																															
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

OCOTP_HW_OCOTP_BOOT_CFG2 field descriptions

Field	Description
BITS	Reflects value of OTP bank 2, word 1. These bits become read-only after the HW_OCOTP_LOCK_BOOT_CFG[1] bit is set.

6.4.19 Value of OTP Bank2 Word2 (Boot Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG3)

Shadowed memory mapped access to OTP bank 2, word 2.

Address: 0h base + 4A0h offset = 4A0h

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	BITS																															
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

OCOTP_HW_OCOTP_BOOT_CFG3 field descriptions

Field	Description
BITS	Reflects value of OTP bank 2, word 2. These bits become read-only after the HW_OCOTP_LOCK_BOOT_CFG[1] bit is set.

6.4.20 Value of OTP Bank2 Word3 (BOOT Configuration Info.) (OCOTP_HW_OCOTP_BOOT_CFG4)

Shadowed memory mapped access to OTP bank 2, word 3.

Address: 0h base + 4B0h offset = 4B0h

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	BITS																															
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

OCOTP_HW_OCOTP_BOOT_CFG4 field descriptions

Field	Description
BITS	Reflects value of OTP bank 2, word 3. These bits become read-only after the HW_OCOTP_LOCK_BOOT_CFG[1] bit is set.

6.4.21 Shadow Register for OTP Bank6 Word0 (SRK Hash) (OCOTP_HW_OCOTP_SRK0)

Shadowed memory mapped access to OTP Bank 6, word 0.

Address: 0h base + 580h offset = 580h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK0 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word0 (Copy of OTP Bank 6, word 0). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.22 Shadow Register for OTP Bank6 Word1 (SRK Hash) (OCOTP_HW_OCOTP_SRK1)

Shadowed memory mapped access to OTP Bank 6, word 1.

Address: 0h base + 590h offset = 590h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK1 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word1 (Copy of OTP Bank 6, word 1). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.23 Shadow Register for OTP Bank6 Word2 (SRK Hash) (OCOTP_HW_OCOTP_SRK2)

Shadowed memory mapped access to OTP Bank 6, word 2.

Address: 0h base + 5A0h offset = 5A0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK2 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word2 (Copy of OTP Bank 6, word 2). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.24 Shadow Register for OTP Bank6 Word3 (SRK Hash) (OCOTP_HW_OCOTP_SRK3)

Shadowed memory mapped access to OTP Bank 6, word 3.

Address: 0h base + 5B0h offset = 5B0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK3 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word3 (Copy of OTP Bank 6, word 3). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.25 Shadow Register for OTP Bank7 Word0 (SRK Hash) (OCOTP_HW_OCOTP_SRK4)

Shadowed memory mapped access to OTP Bank 7, word 0.

Address: 0h base + 5C0h offset = 5C0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK4 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word4 (Copy of OTP Bank 7, word 0). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.26 Shadow Register for OTP Bank7 Word1 (SRK Hash) (OCOTP_HW_OCOTP_SRK5)

Shadowed memory mapped access to OTP Bank 7, word 1.

Address: 0h base + 5D0h offset = 5D0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK5 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word5 (Copy of OTP Bank 7, word 1). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.27 Shadow Register for OTP Bank7 Word2 (SRK Hash) (OCOTP_HW_OCOTP_SRK6)

Shadowed memory mapped access to OTP Bank 7, word 2.

Address: 0h base + 5E0h offset = 5E0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK6 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word6 (Copy of OTP Bank 7, word 2). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.28 Shadow Register for OTP Bank7 Word3 (SRK Hash) (OCOTP_HW_OCOTP_SRK7)

Shadowed memory mapped access to OTP Bank 7, word 3.

Address: 0h base + 5F0h offset = 5F0h

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	BITS																															
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

OCOTP_HW_OCOTP_SRK7 field descriptions

Field	Description
BITS	Shadow register for the hash of the Super Root Key word7 (Copy of OTP Bank 7, word 3). These bits become read-only after the HW_OCOTP_LOCK_SRK bit is set.

6.4.29 Value of OTP Bank8 Word0 (Secure JTAG Response Field) (OCOTP_HW_OCOTP_SJC_RESP0)

Shadowed memory mapped access to OTP Bank 8, word 0.

Address: 0h base + 600h offset = 600h

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	BITS																															
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

OCOTP_HW_OCOTP_SJC_RESP0 field descriptions

Field	Description
BITS	Shadow register for the SJC_RESP Key word0 (Copy of OTP Bank 8, word 0). These bits can be not read and wrotten after the HW_OCOTP_LOCK_SJC_RESP bit is set. If read, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR].

6.4.30 Value of OTP Bank8 Word1 (Secure JTAG Response Field) (OCOTP_HW_OCOTP_SJC_RESP1)

Shadowed memory mapped access to OTP Bank 8, word 1.

Address: 0h base + 610h offset = 610h

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	BITS																															
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

OCOTP_HW_OCOTP_SJC_RESP1 field descriptions

Field	Description
BITS	Shadow register for the SJC_RESP Key word1 (Copy of OTP Bank 8, word 1). These bits can be not read and wrotten after the HW_OCOTP_LOCK_SJC_RESP bit is set. If read, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR].

6.4.31 Value of OTP Bank8 Word2 (USB ID info) (OCOTP_HW_OCOTP_USB_ID)

Shadowed memory mapped access to OTP Bank 8, word 2.

Address: 0h base + 620h offset = 620h

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	BITS																															
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

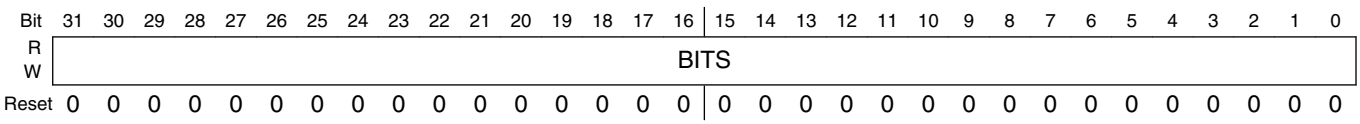
OCOTP_HW_OCOTP_USB_ID field descriptions

Field	Description
BITS	Reflects value of OTP Bank 8, word 2.

6.4.32 Value of OTP Bank8 Word3 (Field Return) (OCOTP_HW_OCOTP_FIELD_RETURN)

Shadowed memory mapped access to OTP Bank 8, word 3.

Address: 0h base + 630h offset = 630h



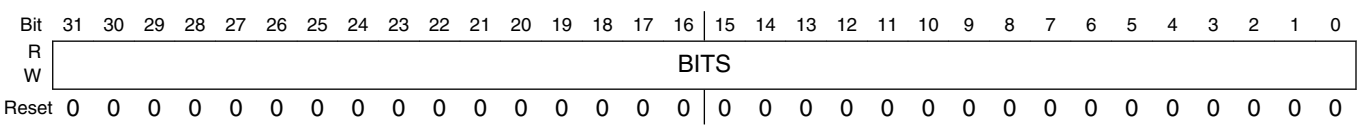
OCOTP_HW_OCOTP_FIELD_RETURN field descriptions

Field	Description
BITS	Reflects value of OTP Bank 8, word 3.

6.4.33 Value of OTP Bank9 Word0 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR0)

Shadowed memory mapped access to OTP Bank 9, word 0.

Address: 0h base + 640h offset = 640h



OCOTP_HW_OCOTP_MAC_ADDR0 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 9, word 0.

6.4.34 Value of OTP Bank9 Word1 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR1)

Shadowed memory mapped access to OTP Bank 9, word 1.

Address: 0h base + 650h offset = 650h

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	BITS																															
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

OCOTP_HW_OCOTP_MAC_ADDR1 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 9, word 1.

6.4.35 Value of OTP Bank9 Word2 (MAC Address) (OCOTP_HW_OCOTP_MAC_ADDR2)

Shadowed memory mapped access to OTP Bank 9, word 2.

Address: 0h base + 660h offset = 660h

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	BITS																															
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

OCOTP_HW_OCOTP_MAC_ADDR2 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 9, word 2.

6.4.36 Value of OTP Bank14 Word0 () (OCOTP_HW_OCOTP_GP10)

Shadowed memory mapped access to OTP Bank 14, word 0.

Address: 0h base + 780h offset = 780h

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	BITS																															
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

OCOTP_HW_OCOTP_GP10 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 14, word 0.

6.4.37 Value of OTP Bank14 Word1 () (OCOTP_HW_OCOTP_GP11)

Shadowed memory mapped access to OTP Bank 14, word 1.

Address: 0h base + 790h offset = 790h

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	BITS																															
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

OCOTP_HW_OCOTP_GP11 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 14, word 1.

6.4.38 Value of OTP Bank14 Word2 () (OCOTP_HW_OCOTP_GP20)

Shadowed memory mapped access to OTP Bank 14, word 2.

Address: 0h base + 7A0h offset = 7A0h

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	BITS																															
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

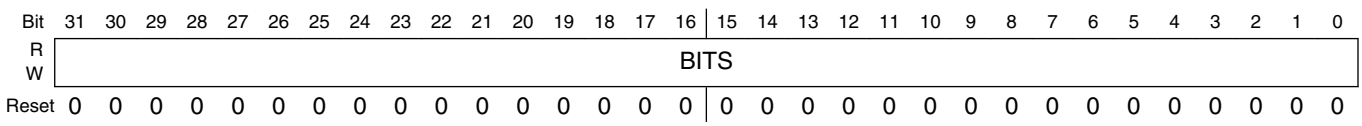
OCOTP_HW_OCOTP_GP20 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 14, word 2.

6.4.39 Value of OTP Bank14 Word3 () (OCOTP_HW_OCOTP_GP21)

Shadowed memory mapped access to OTP Bank 14, word 3.

Address: 0h base + 7B0h offset = 7B0h



OCOTP_HW_OCOTP_GP21 field descriptions

Field	Description
BITS	Reflects value of OTP Bank 14, word 3.

Chapter 7
Central Security Unit (CSU)

7.1 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.

7.1.1 Features

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

The CSU has these security-related features:

- Peripheral access policy—the appropriate bus master privilege and identity are required to access each peripheral.
- Masters privilege policy—the CSU overrides the bus master privilege signals (user/supervisor, secure/non-secure) according to the access control policy.

7.2 Functional description

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

The security policies may be set, and optionally locked in the CSU registers. The examples of a secure software include the Command Sequence File (CSF) processed by the High Assurance Boot (HAB) or the HAB-authenticated image which executes after the boot ROM.

7.2.1 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 (bus privileges) in the system distinguished by the security (TrustZone/non-TrustZone) and privilege (Supervisor/User) setting of the module. This is the list of these security modes, organized 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 the master privileges that can access the peripheral. The relationship between the value of the csu_sec_level signal and the security operation mode is shown in this table:

Table 7-1. Access permissions

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

7.2.2 Initialization policy

The recommended initialization procedure is as follows:

1. Write the CSU_CSL register field value to indicate each peripheral's privilege mode.
2. Write the HP register field value to override the master's privilege mode.

NOTE

After programming, the register lock bit must be set to prevent further modifications to a register value.

7.3 Programmable Registers

The following sections provide a detailed description 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 the secure supervisor mode. (Note: These registers are also referred to as the 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
303E_0000	Config security level register (CSU_CSL0)	32	R/W	0033_0033h	7.3.1/203
303E_0004	Config security level register (CSU_CSL1)	32	R/W	0033_0033h	7.3.1/203
303E_0008	Config security level register (CSU_CSL2)	32	R/W	0033_0033h	7.3.1/203
303E_000C	Config security level register (CSU_CSL3)	32	R/W	0033_0033h	7.3.1/203
303E_0010	Config security level register (CSU_CSL4)	32	R/W	0033_0033h	7.3.1/203
303E_0014	Config security level register (CSU_CSL5)	32	R/W	0033_0033h	7.3.1/203
303E_0018	Config security level register (CSU_CSL6)	32	R/W	0033_0033h	7.3.1/203
303E_001C	Config security level register (CSU_CSL7)	32	R/W	0033_0033h	7.3.1/203
303E_0020	Config security level register (CSU_CSL8)	32	R/W	0033_0033h	7.3.1/203
303E_0024	Config security level register (CSU_CSL9)	32	R/W	0033_0033h	7.3.1/203
303E_0028	Config security level register (CSU_CSL10)	32	R/W	0033_0033h	7.3.1/203
303E_002C	Config security level register (CSU_CSL11)	32	R/W	0033_0033h	7.3.1/203
303E_0030	Config security level register (CSU_CSL12)	32	R/W	0033_0033h	7.3.1/203

Table continues on the next page...

CSU memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303E_0034	Config security level register (CSU_CSL13)	32	R/W	0033_0033h	7.3.1/203
303E_0038	Config security level register (CSU_CSL14)	32	R/W	0033_0033h	7.3.1/203
303E_003C	Config security level register (CSU_CSL15)	32	R/W	0033_0033h	7.3.1/203
303E_0040	Config security level register (CSU_CSL16)	32	R/W	0033_0033h	7.3.1/203
303E_0044	Config security level register (CSU_CSL17)	32	R/W	0033_0033h	7.3.1/203
303E_0048	Config security level register (CSU_CSL18)	32	R/W	0033_0033h	7.3.1/203
303E_004C	Config security level register (CSU_CSL19)	32	R/W	0033_0033h	7.3.1/203
303E_0050	Config security level register (CSU_CSL20)	32	R/W	0033_0033h	7.3.1/203
303E_0054	Config security level register (CSU_CSL21)	32	R/W	0033_0033h	7.3.1/203
303E_0058	Config security level register (CSU_CSL22)	32	R/W	0033_0033h	7.3.1/203
303E_005C	Config security level register (CSU_CSL23)	32	R/W	0033_0033h	7.3.1/203
303E_0060	Config security level register (CSU_CSL24)	32	R/W	0033_0033h	7.3.1/203
303E_0064	Config security level register (CSU_CSL25)	32	R/W	0033_0033h	7.3.1/203
303E_0068	Config security level register (CSU_CSL26)	32	R/W	0033_0033h	7.3.1/203
303E_006C	Config security level register (CSU_CSL27)	32	R/W	0033_0033h	7.3.1/203
303E_0070	Config security level register (CSU_CSL28)	32	R/W	0033_0033h	7.3.1/203
303E_0074	Config security level register (CSU_CSL29)	32	R/W	0033_0033h	7.3.1/203
303E_0078	Config security level register (CSU_CSL30)	32	R/W	0033_0033h	7.3.1/203
303E_007C	Config security level register (CSU_CSL31)	32	R/W	0033_0033h	7.3.1/203
303E_0080	Config security level register (CSU_CSL32)	32	R/W	0033_0033h	7.3.1/203
303E_0084	Config security level register (CSU_CSL33)	32	R/W	0033_0033h	7.3.1/203
303E_0088	Config security level register (CSU_CSL34)	32	R/W	0033_0033h	7.3.1/203
303E_008C	Config security level register (CSU_CSL35)	32	R/W	0033_0033h	7.3.1/203
303E_0090	Config security level register (CSU_CSL36)	32	R/W	0033_0033h	7.3.1/203
303E_0094	Config security level register (CSU_CSL37)	32	R/W	0033_0033h	7.3.1/203
303E_0098	Config security level register (CSU_CSL38)	32	R/W	0033_0033h	7.3.1/203
303E_009C	Config security level register (CSU_CSL39)	32	R/W	0033_0033h	7.3.1/203
303E_00A0	Config security level register (CSU_CSL40)	32	R/W	0033_0033h	7.3.1/203
303E_00A4	Config security level register (CSU_CSL41)	32	R/W	0033_0033h	7.3.1/203
303E_00A8	Config security level register (CSU_CSL42)	32	R/W	0033_0033h	7.3.1/203
303E_00AC	Config security level register (CSU_CSL43)	32	R/W	0033_0033h	7.3.1/203
303E_00B0	Config security level register (CSU_CSL44)	32	R/W	0033_0033h	7.3.1/203
303E_00B4	Config security level register (CSU_CSL45)	32	R/W	0033_0033h	7.3.1/203
303E_00B8	Config security level register (CSU_CSL46)	32	R/W	0033_0033h	7.3.1/203
303E_00BC	Config security level register (CSU_CSL47)	32	R/W	0033_0033h	7.3.1/203
303E_00C0	Config security level register (CSU_CSL48)	32	R/W	0033_0033h	7.3.1/203
303E_00C4	Config security level register (CSU_CSL49)	32	R/W	0033_0033h	7.3.1/203
303E_00C8	Config security level register (CSU_CSL50)	32	R/W	0033_0033h	7.3.1/203

Table continues on the next page...

CSU memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303E_00CC	Config security level register (CSU_CSL51)	32	R/W	0033_0033h	7.3.1/203
303E_00D0	Config security level register (CSU_CSL52)	32	R/W	0033_0033h	7.3.1/203
303E_00D4	Config security level register (CSU_CSL53)	32	R/W	0033_0033h	7.3.1/203
303E_00D8	Config security level register (CSU_CSL54)	32	R/W	0033_0033h	7.3.1/203
303E_00DC	Config security level register (CSU_CSL55)	32	R/W	0033_0033h	7.3.1/203
303E_00E0	Config security level register (CSU_CSL56)	32	R/W	0033_0033h	7.3.1/203
303E_00E4	Config security level register (CSU_CSL57)	32	R/W	0033_0033h	7.3.1/203
303E_00E8	Config security level register (CSU_CSL58)	32	R/W	0033_0033h	7.3.1/203
303E_00EC	Config security level register (CSU_CSL59)	32	R/W	0033_0033h	7.3.1/203
303E_00F0	Config security level register (CSU_CSL60)	32	R/W	0033_0033h	7.3.1/203
303E_00F4	Config security level register (CSU_CSL61)	32	R/W	0033_0033h	7.3.1/203
303E_00F8	Config security level register (CSU_CSL62)	32	R/W	0033_0033h	7.3.1/203
303E_00FC	Config security level register (CSU_CSL63)	32	R/W	0033_0033h	7.3.1/203
303E_0200	HP0 register (CSU_HP0)	32	R/W	0000_0000h	7.3.2/207
303E_0204	HP1 register (CSU_HP1)	32	R/W	0000_0000h	7.3.3/211
303E_0218	Secure access register (CSU_SA)	32	R/W	0000_0000h	7.3.4/211
303E_0358	HPCONTROL0 register (CSU_HPCONTROL0)	32	R/W	0000_0000h	7.3.5/215
303E_035C	HPCONTROL1 register (CSU_HPCONTROL1)	32	R/W	0000_0000h	7.3.6/219

7.3.1 Config security level register (CSU_CSL*n*)

There are several config security level (CSU_CSL0-CSU_CSL*n*) registers. Each CSU_CSL comprises of two fields, with 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 located in b23-b16 and bits b7-b0, respectively.

The permission access table Table 7-1 shows the security levels and the 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 the allocation of the CSL register per slave or a group of slave modules.

CSLn register	CSL[2n] field (index)	Slave module	CSL[2n+1] field (index)	Slave module
0	0	GPIO1	1	GPIO2
1	2	GPIO3	3	GPIO4

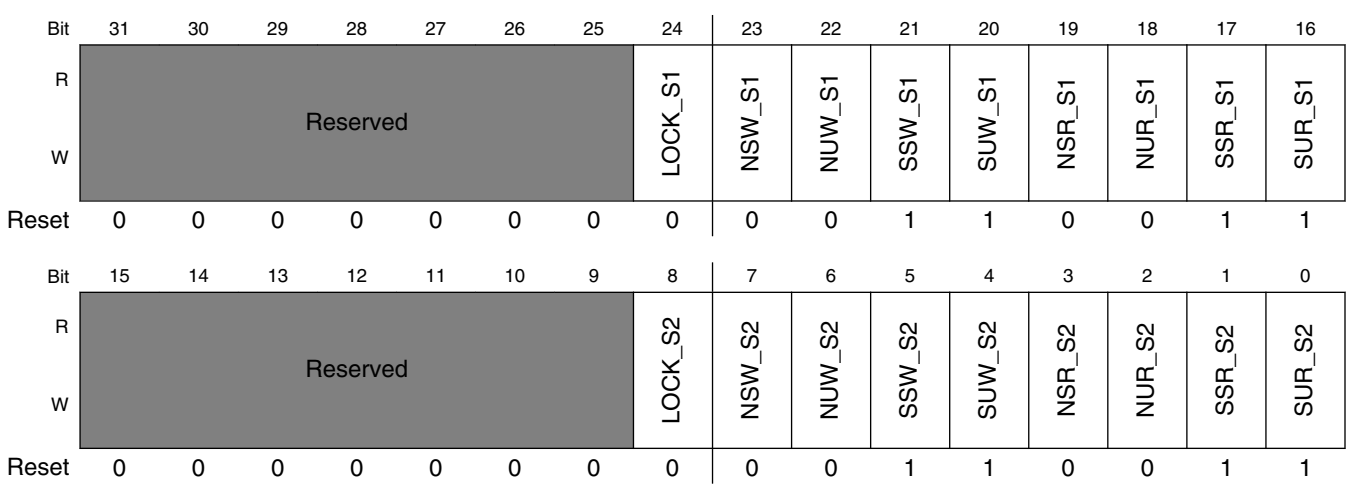
Table continues on the next page...

CSLn register	CSL[2n] field (index)	Slave module	CSL[2n+1] field (index)	Slave module
2	4	GPIO5	5	Reserved
3	6	ANA_TSENSOR	7	ANA_OSC
4	8	WDOG1	9	WDOG2
5	10	WDOG3	11	Reserved
6	12	SDMA2	13	GPT1
7	14	GPT2	15	GPT3
8	16	Reserved	17	ROMCP
9	18	LCDIF	19	IOMUXC
10	20	IOMUXC_GPR	21	OCOTP_CTRL
11	22	ANA_PLL	23	SNVS_HP
12	24	CCM	25	SRC
13	26	GPC	27	SEMAPHORE1
14	28	SEMAPHORE2	29	RDC
15	30	CSU	31	Reserved
16	32	DC_MST0	33	DC_MST1
17	34	DC_MST2	35	DC_MST3
18	36	Reserved	37	Reserved
19	38	PWM1	39	PWM2
20	40	PWM3	41	PWM4
21	42	System_Counter_RD	43	System_Counter_CMP
22	44	System_Counter_CTRL	45	Reserved
23	46	GPT6	47	GPT5
24	48	GPT4	49	Reserved
25	50	Reserved	51	Reserved
26	52	Reserved	53	Reserved
27	54	Reserved	55	Reserved
28	56	TZASC	57	Reserved
29	58	Reserved	59	MTR
30	60	PERFMON1	61	PERFMON2
31	62	PLATFORM_CTRL	63	QoSC
32	64	MIPI_PHY	65	MIPI_DSI
33	66	I2C1	67	I2C2
34	68	I2C3	69	I2C4
35	70	UART4	71	MIPI_CSI1
36	72	MIPI_CSI_PHY1	73	CSI1
37	74	MU_A	75	MU_B
38	76	SEMAPHORE_HS	77	Reserved for SDMA2 internal memory
39	78	SAI1	79	Reserved
40	80	SAI6	81	SAI5

Table continues on the next page...

CSLn register	CSL[2n] field (index)	Slave module	CSL[2n+1] field (index)	Slave module
41	82	SAI4	83	Reserved for SDMA2 internal memory
42	84	uSDHC1	85	uSDHC2
43	86	MIPI_CSI2	87	MIPI_CSI_PHY2
44	88	CSI2	89	Reserved for SDMA2 internal memory
45	90	SPBA2	91	QSPI
46	92	Reserved	93	SDMA1
47	94	ENET1	95	Reserved
48	96	Reserved for SDMA internal memory	97	SPDIF1
49	98	eCSPI1	99	eCSPI2
50	100	eCSPI3	101	Reserved
51	102	UART1	103	Reserved for SDMA internal registers
52	104	UART3	105	UART2
53	106	SPDIF2	107	SAI2
54	108	SAI3	109	Reserved
55	110	Reserved for SDMA internal registers	111	SPBA1
56	112	module_en_glbl[0]	113	module_en_glbl[0]
57	114	CAAM	115	Reserved
58	116	Reserved	117	Reserved
59	118	Reserved	119	Reserved
60	120	Reserved	121	Reserved
61	122	Reserved	123	Reserved
62	124	Reserved	125	Reserved
63	126	Reserved	127	Reserved

Address: 303E_0000h base + 0h offset + (4d × i), where i=0d to 63d



CSU_CSL*n* field descriptions

Field	Description
31–25 -	This field is reserved. Reserved.
24 LOCK_S1	The lock bit corresponding to the first slave. It is written by the secure software. 0 Not locked. The bits 16-23 can be written by the software. 1 The bits 16-23 are locked and can't be written by the software.
23 NSW_S1	Non-secure supervisor write access control for the first slave 0 The non-secure supervisor write access is disabled for the first slave. 1 The non-secure supervisor write access is enabled for the first slave
22 NUW_S1	Non-secure user write access control for the first slave 0 The non-secure user write access is disabled for the first slave. 1 The non-secure user write access is enabled for the first slave.
21 SSW_S1	Secure supervisor write access control for the first slave 0 The secure supervisor write access is disabled for the first slave. 1 The secure supervisor write access is enabled for the first slave.
20 SUW_S1	Secure user write access control for the first slave 0 The secure user write access is disabled for the first slave. 1 The secure user write access is enabled for the first slave.
19 NSR_S1	Non-secure supervisor read access control for the first slave 0 The non-secure supervisor read access is disabled for the first slave. 1 The non-secure supervisor read access is enabled for the first slave.
18 NUR_S1	Non-secure user read access control for the first slave 0 The non-secure user read access is disabled for the first slave. 1 The non-secure user read access is enabled for the first slave.
17 SSR_S1	Secure supervisor read access control for the first slave 0 The secure supervisor read access is disabled for the first slave. 1 The secure supervisor read access is enabled for the first slave.
16 SUR_S1	Secure user read access control for the first slave 0 The secure user read access is disabled for the first slave. 1 The secure user read access is enabled for the first slave.
15–9 -	This field is reserved. Reserved
8 LOCK_S2	The lock bit corresponding to the second slave. It is written by the secure software. 0 Not locked. Bits 7-0 can be written by the software. 1 Bits 7-0 are locked and cannot be written by the software
7 NSW_S2	Non-secure supervisor write access control for the second slave 0 The non-secure supervisor write access is disabled for the second slave. 1 The non-secure supervisor write access is enabled for the second slave.

Table continues on the next page...

CSU_CSL*n* field descriptions (continued)

Field	Description
6 NUW_S2	Non-secure user write access control for the second slave 0 The non-secure user write access is disabled for the second slave. 1 The non-secure user write access is enabled for the second slave.
5 SSW_S2	Secure supervisor write access control for the second slave 0 The secure supervisor write access is disabled for the second slave. 1 The secure supervisor write access is enabled for the second slave.
4 SUW_S2	Secure user write access control for the second slave 0 The secure user write access is disabled for the second slave. 1 The secure user write access is enabled for the second slave.
3 NSR_S2	Non-secure supervisor read access control for the second slave 0 The non-secure supervisor read access is disabled for the second slave. 1 The non-secure supervisor read access is enabled for the second slave.
2 NUR_S2	Non-secure user read access control for the second slave 0 The non-secure user read access is disabled for the second slave. 1 The non-secure user read access is enabled for the second slave.
1 SSR_S2	Secure supervisor read access control for the second slave 0 The secure supervisor read access is disabled for the second slave. 1 The secure supervisor read access is enabled for the second slave.
0 SUR_S2	Secure user read access control for the second slave 0 The secure user read access is disabled for the second slave. 1 The secure user read access is enabled for the second slave.

7.3.2 HP0 register (CSU_HP0)

The SCU_HP0 and SCU_HP1 registers can be programmed to determine the privilege (either the user mode or the supervisor mode) for 17 different master groups. The privilege of the particular master group can 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 the lock bits which enable/disable writing to the corresponding privilege indicator bits.

Address: 303E_0000h base + 200h offset = 303E_0200h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	L_SDMA2	HP_SDMA2	L_DAP	HP_DAP	L_HUGO	HP_HUGO	L_USDHC3	HP_USDHC3	L_USDHC2	HP_USDHC2	L_USDHC1	HP_USDHC1	L_ENET1	HP_ENET1	L_APBHDMA_RAWNAND	HP_APBHDMA_RAWNAND
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	L_GPU	HP_GPU	L_VPU_Decoders	HP_VPU_Decoders	L_TPSMP_PCIE_CTRL1_CTRL2	HP_TPSMP_PCIE_CTRL1_CTRL2	L_USB1_USB2	HP_USB1_USB2	L_LCDIF_CS1_CS2	HP_LCDIF_CS1_CS2	L_SDMA1	HP_SDMA1	L_M4	HP_M4	Reserved	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CSU_HP0 field descriptions

Field	Description
31 L_SDMA2	Lock bit set by the TZ software for the HP_SDMA2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
30 HP_SDMA2	Indicates the privilege/user mode for the SDMA2 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
29 L_DAP	Lock bit set by the TZ software for the HP_DAP 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
28 HP_DAP	Indicates the privilege/user mode for the DAP 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
27 L_HUGO	Lock bit set by the TZ software for the HP_HUGO 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
26 HP_HUGO	Indicates the privilege/user mode for the HUGO 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
25 L_USDHC3	Lock bit set by the TZ software for the HP_USDHC3

Table continues on the next page...

CSU_HP0 field descriptions (continued)

Field	Description
	0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
24 HP_USDHC3	Indicates the privilege/user mode for the USDHC3 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
23 L_USDHC2	Lock bit set by the TZ software for the HP_USDHC2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
22 HP_USDHC2	Indicates the privilege/user mode for the USDHC2 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
21 L_USDHC1	Lock bit set by the TZ software for the HP_USDHC1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
20 HP_USDHC1	Indicates the privilege/user mode for the USDHC1 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
19 L_ENET1	Lock bit set by the TZ software for the HP_ENET1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
18 HP_ENET1	Indicates the privilege/user mode for the ENET 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
17 L_APBHDMA_ RAWNAND	Lock bit set by the TZ software for the HP_APBHDMA_RAWNAND 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
16 HP_APBHDMA_ RAWNAND	Indicates the privilege/user mode for the RawNAND 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
15 L_GPU	Lock bit set by the TZ software for the HP_GPU 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
14 HP_GPU	Indicates the privilege/user mode for the GPU 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
13 L_VPU_ Decoders	Lock bit set by the TZ software for the HP_VPU_Decoders

Table continues on the next page...

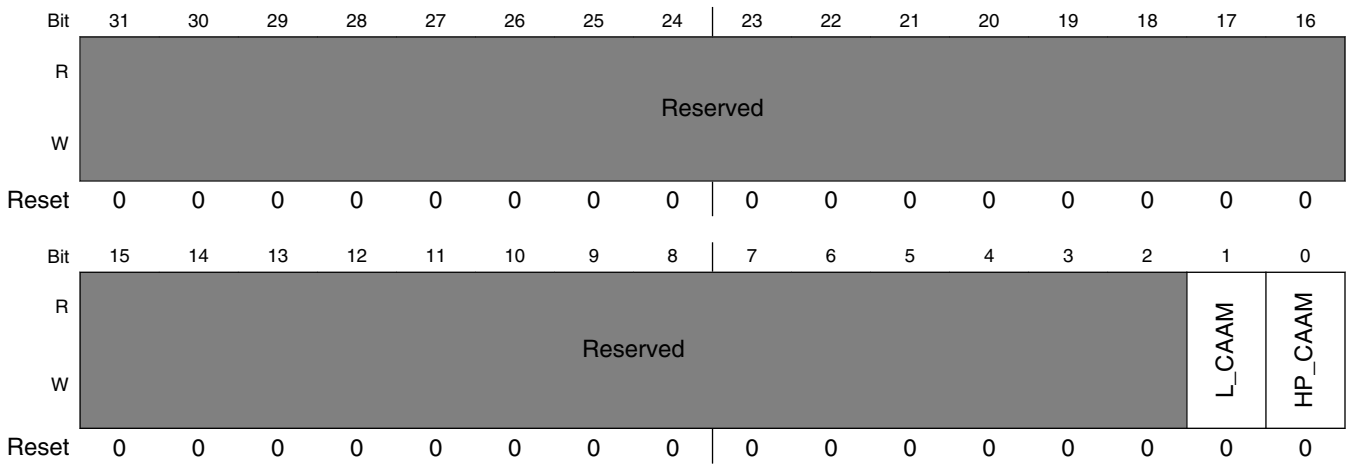
CSU_HP0 field descriptions (continued)

Field	Description
	0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
12 HP_VPU_ Decoders	Indicates the privilege/user mode for the VPU 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
11 L_TPSMP_ PCIE_CTRL1_ CTRL2	Lock bit set by the TZ software for the HP_TPSMP_PCIE_CTRL1_CTRL2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
10 HP_TPSMP_ PCIE_CTRL1_ CTRL2	Indicates the privilege/user mode for the PCIE 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
9 L_USB1_USB2	Lock bit set by the TZ software for the HP_USB1_USB2. 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
8 HP_USB1_USB2	Indicates the privilege/user mode for the USB 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
7 L_LCDIF_CS1_ CS2	Lock bit set by the TZ software for the HP_LCDIF_CS1_CS2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
6 HP_LCDIF_CS1_ CS2	Indicates the privilege/user mode for the LCDIF 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
5 L_SDMA1	Lock bit set by the TZ software for the HP_SDMA1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
4 HP_SDMA1	Indicates the privilege/user mode for the SDMA1 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
3 L_M4	Lock bit set by the TZ software for the HP_M4 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
2 HP_M4	Indicates the privilege/user mode for the M4 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master
-	This field is reserved.

7.3.3 HP1 register (CSU_HP1)

The SCU_HP1 register is an expansion of the SCU_HP0 register. See the SCU_HP0 register definition.

Address: 303E_0000h base + 204h offset = 303E_0204h



CSU_HP1 field descriptions

Field	Description
31–2 -	This field is reserved. Reserved
1 L_CAAM	Lock bit set by the TZ software for the CAAM 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
0 HP_CAAM	Indicates the privilege/user mode for the CAAM 0 User mode for the corresponding master 1 Supervisor mode for the corresponding master

7.3.4 Secure access register (CSU_SA)

The secure access register can be programmed to specify the access policy (either secure or non-secure) for up to 16 different masters. This register is used to set the access policy for the type-1 masters which are not capable of setting the policy by themselves.

The 16 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: 303E_0000h base + 218h offset = 303E_0218h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	L_SDMA2	NSA_SDMA2	L_DAP	NSA_DAP	L_HUGO	NSA_HUGO	L_USDHC3	NSA_USDHC3	L_USDHC2	NSA_USDHC2	L_USDHC1	NSA_USDHC1	L_ENET1	NSA_ENET1	L_APBHDMA_RAWNAND	NSA_APBHDMA_RAWNAND
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	L_GPU	NSA_GPU	L_VPU_Decoders	NSA_VPU_Decoders	L_TPSMP_PCIE_CTRL1_CTRL2	NSA_TPSMP_PCIE_CTRL1_CTRL2	L_USB1_USB2	NSA_USB1_USB2	L_LCDIF_CS1_CS2	NSA_LCDIF_CS1_CS2	L_SDMA1	NSA_SDMA1	L_M4	NSA_M4	Reserved	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CSU_SA field descriptions

Field	Description
31 L_SDMA2	Lock bit set by the TZ software for the NSA_SDMA2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
30 NSA_SDMA2	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the SDMA2 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
29 L_DAP	Lock bit set by the TZ software for the NSA_DAP 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
28 NSA_DAP	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the DAP 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
27 L_HUGO	Lock bit set by the TZ software for the NSA_HUGO 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
26 NSA_HUGO	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the HUGO

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
25 L_USDHC3	Lock bit set by the TZ software for the NSA_USDHC3 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
24 NSA_USDHC3	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the 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 the TZ software for the NSA_USDHC2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
22 NSA_USDHC2	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the 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 the TZ software for the NSA_USDHC1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
20 NSA_USDHC1	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the USDHC1 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
19 L_ENET1	Lock bit set by the TZ software for the NSA_ENET1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
18 NSA_ENET1	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the ENET 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
17 L_APBHDMA_ RAWNAND	Lock bit set by the TZ software for the NSA_APBHDMA_RAWNAND 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
16 NSA_ APBHDMA_ RAWNAND	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the RawNAND

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
15 L_GPU	Lock bit set by the TZ software for the NSA_GPU 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
14 NSA_GPU	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the GPU 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
13 L_VPU_ Decoders	Lock bit set by the TZ software for the NSA_VPU_Decoders 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
12 NSA_VPU_ Decoders	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the VPU 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
11 L_TPSMP_ PCIE_CTRL1_ CTRL2	Lock bit set by the TZ software for the NSA_TPSMP_PCIE_CTRL1_CTRL2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
10 NSA_TPSMP_ PCIE_CTRL1_ CTRL2	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the PCIE 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
9 L_USB1_USB2	Lock bit set by the TZ software for the NSA_USB1_USB2. 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
8 NSA_USB1_ USB2	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the USB 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
7 L_LCDIF_CS1_ CS2	Lock bit set by the TZ software for the NSA_LCDIF_CS1_CS2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
6 NSA_LCDIF_ CS1_CS2	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the LCDIF

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
5 L_SDMA1	Lock bit set by the TZ software for the NSA_SDMA1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
4 NSA_SDMA1	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the SDMA1 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
3 L_M4	Lock bit set by the TZ software for the NSA_M4 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
2 NSA_M4	Non-secure access policy indicator bit Indicates the access type (secure/non-secure) to the M4 0 Secure access for the corresponding type-1 master 1 Non-secure access for the corresponding type-1 master
-	This field is reserved.

7.3.5 HPCONTROL0 register (CSU_HPCONTROL0)

The HP control registers CSU_HPCONTROL0 and CSU_HPCONTROL1 enable the CSU to control the USER/SUPERVISOR mode state for the specified masters. The register toggles the csu_hprot1 output signal for the system masters. The two possible sources for the csu_hprot1 output are:

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

The even bits in the registers are used to lock the control bit values.

Address: 303E_0000h base + 358h offset = 303E_0358h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	L_SDMA2	HPC_SDMA2	L_DAP	HPC_DAP	L_HUGO	HPC_HUGO	L_USDHC3	HPC_USDHC3	L_USDHC2	HPC_USDHC2	L_USDHC1	HPC_USDHC1	L_ENET1	HPC_ENET1	L_APBHDMA_RAWNAND	HPC_APBHDMA_RAWNAND
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	L_GPU	HPC_GPU	L_VPU_Decoders	HPC_VPU_Decoders	L_TPSMP_PCIE_CTRL1_CTRL2	HPC_TPSMP_PCIE_CTRL1_CTRL2	L_USB1_USB2	HPC_USB1_USB2	L_LCDIF_CS1_CS2	HPC_LCDIF_CS1_CS2	L_SDMA1	HPC_SDMA1	L_M4	HPC_M4	Reserved	
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_SDMA2	Lock bit set by the TZ software for the HPC_SDMA2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
30 HPC_SDMA2	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the SDMA2 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
29 L_DAP	Lock bit set by the TZ software for the HPC_DAP 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
28 HPC_DAP	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the DAP 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
27 L_HUGO	Lock bit set by the TZ software for the HPC_HUGO 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
26 HPC_HUGO	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the HUGO 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.

Table continues on the next page...

CSU_HPCONTROL0 field descriptions (continued)

Field	Description
25 L_USDHC3	Lock bit set by the TZ software for the HPC_USDHC3 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
24 HPC_USDHC3	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the USDHC3 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
23 L_USDHC2	Lock bit set by the TZ software for the HPC_USDHC2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
22 HPC_USDHC2	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the USDHC2 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
21 L_USDHC1	Lock bit set by the TZ software for the HPC_USDHC1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
20 HPC_USDHC1	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the USDHC1 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
19 L_ENET1	Lock bit set by the TZ software for the HPC_ENET1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
18 HPC_ENET1	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the ENET 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
17 L_APBHDMA_ RAWNAND	Lock bit set by the TZ software for the HPC_APBHDMA_RAWNAND 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
16 HPC_ APBHDMA_ RAWNAND	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of thee RawNAND 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
15 L_GPU	Lock bit set by the TZ software for the HPC_GPU 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
14 HPC_GPU	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the GPU

Table continues on the next page...

CSU_HPCONTROL0 field descriptions (continued)

Field	Description
	0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
13 L_VPU_ Decoders	Lock bit set by the TZ software for the HPC_VPU_Decoders 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
12 HPC_VPU_ Decoders	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the VPU 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
11 L_TPSMP_ PCIE_CTRL1_ CTRL2	Lock bit set by the TZ software for the HPC_TPSMP_PCIE_CTRL1_CTRL2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
10 HPC_TPSMP_ PCIE_CTRL1_ CTRL2	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the PCIE 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
9 L_USB1_USB2	Lock bit set by the TZ software for the HPC_USB1_USB2. 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
8 HPC_USB1_ USB2	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the USB 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
7 L_LCDIF_CS1_ CS2	Lock bit set by the TZ software for the HPC_LCDIF_CS1_CS2 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
6 HPC_LCDIF_ CS1_CS2	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the LCDIF 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
5 L_SDMA1	Lock bit set by the TZ software for the HPC_SDMA1 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
4 HPC_SDMA1	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the SDMA1 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
3 L_M4	Lock bit set by the TZ software for the HPC_M4 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.

Table continues on the next page...

CSU_HPCONTROL0 field descriptions (continued)

Field	Description
2 HPC_M4	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the M4 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.
-	This field is reserved.

7.3.6 HPCONTROL1 register (CSU_HPCONTROL1)

The SCU_HPCONTROL1 register is an expansion of the SCU_HPCONTROL0 register. See the SCU_HPCONTROL0 register definition.

Address: 303E_0000h base + 35Ch offset = 303E_035Ch

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	-														L_CAAM	HPC_CAAM
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_CAAM	Lock bit set by the TZ software for the CAAM. 0 No lock—the adjacent (next lower) bit can be written by the software. 1 Lock—the adjacent (next lower) bit can't be written by the software.
0 HPC_CAAM	Determines whether the register value of the corresponding HP field is passed as the hprot[1] of the CAAM. 0 The hprot1 input signal value is routed to the csu_hprot1 output for the corresponding master. 1 The HP register bit is routed to the csu_hprot1 output for the corresponding master.

Chapter 8
Resource Domain Controller (RDC)

8.1 Overview

The Resource Domain Controller (RDC) provides robust support for the isolation of destination memory mapped locations such as peripherals and memory to a single core, a bus master, or set of cores and bus masters.

Many of today's processors have multiple cores for increased performance and flexibility. In some cases, the cores serve different functions (e.g. user level applications versus real time machine control) and in such cases the software for each core may be developed by different providers.

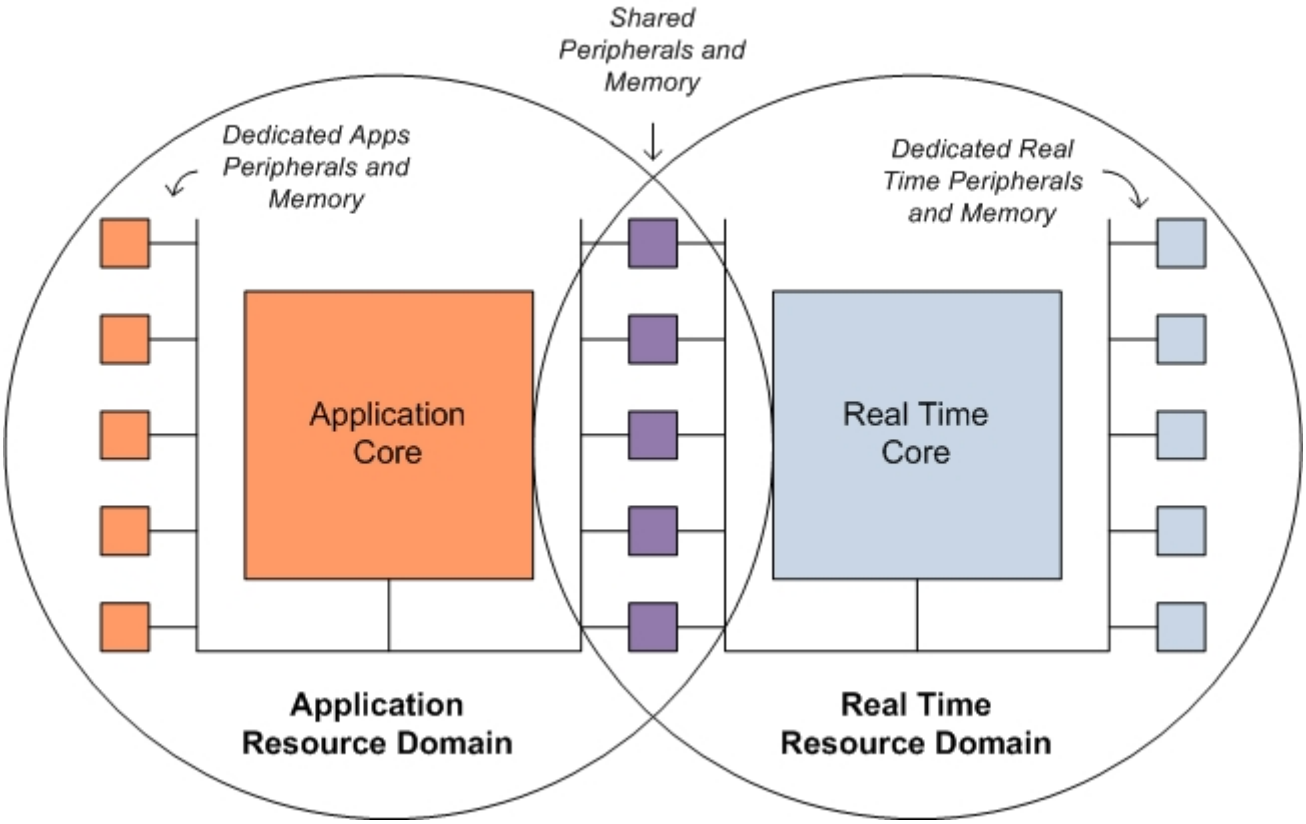


Figure 8-1. Dedicated and Shared Peripherals

For efficiency reasons the code on the cores may share chip resources such as peripherals and memory. The sharing of chip resources between the somewhat independent processing domains allows for the opportunity of data collisions where information stored in peripherals or memory by a process on one core is overwritten by software running on another core. Without careful collaboration between the two operating systems inadvertent malfunction or degradation in performance may result.

The RDC provides a mechanism to allow boot time configuration code to establish resource domains by assigning cores, bus masters, peripherals and memory regions to domain identifiers. Once configured, bus transactions are monitored to restrict accesses initiated by cores and bus masters to their respective peripherals and memory.

For shared peripherals, the RDC provides a semaphore-based locking mechanism to provide for temporary exclusivity while the domain software uses the peripheral. Once the software of one domain has finished the task and finished with the peripheral then it may release the semaphore making the peripheral available to the other domain.

8.1.1 Features

Resource domain subsystem has the following features:

- Assignment of cores, bus masters, peripherals, and memory regions to a resource domain
- Fixed memory resolution of 128 Bytes for small address spaces and 4 KB for large address spaces
- Four resource domain identifiers
- Memory read/write access controls for each resource domain and region
- Optional semaphore-based, hardware-enforced exclusive access of shared peripherals to a resource domain
- Prioritized access permissions for overlapping memory regions
- Automatic restoration of resource domain access permissions to memory regions in the power-down domain

8.2 Functional Description

The RDC is the central location for creation of isolated resource domains and for the enablement of semaphore-based access also known as “safe sharing”. Configuration software assigns one of four resource domain identifiers to each core and bus master, and allocates each memory region and peripheral to one or more resource domains.

Memory Read or Write access privileges for each resource domain are declared for each memory region. In addition, the software configuration determines which shared peripherals (those peripherals allocated to more than one domain) require safe sharing by setting the semaphore-required configuration for each peripheral.

The RDC configuration information is sent to the fabric ports, memories gaskets, semaphore controller, and peripherals to control access based on domain assignments. The fabric uses the domain identifier associated with each port to include this information along with the bus transaction. When the slave gasket encounters a bus transaction it makes a comparison of the transaction domain ID to the RDC-provided list of allowed domains. If the transaction domain ID is on the list then access may be permitted.

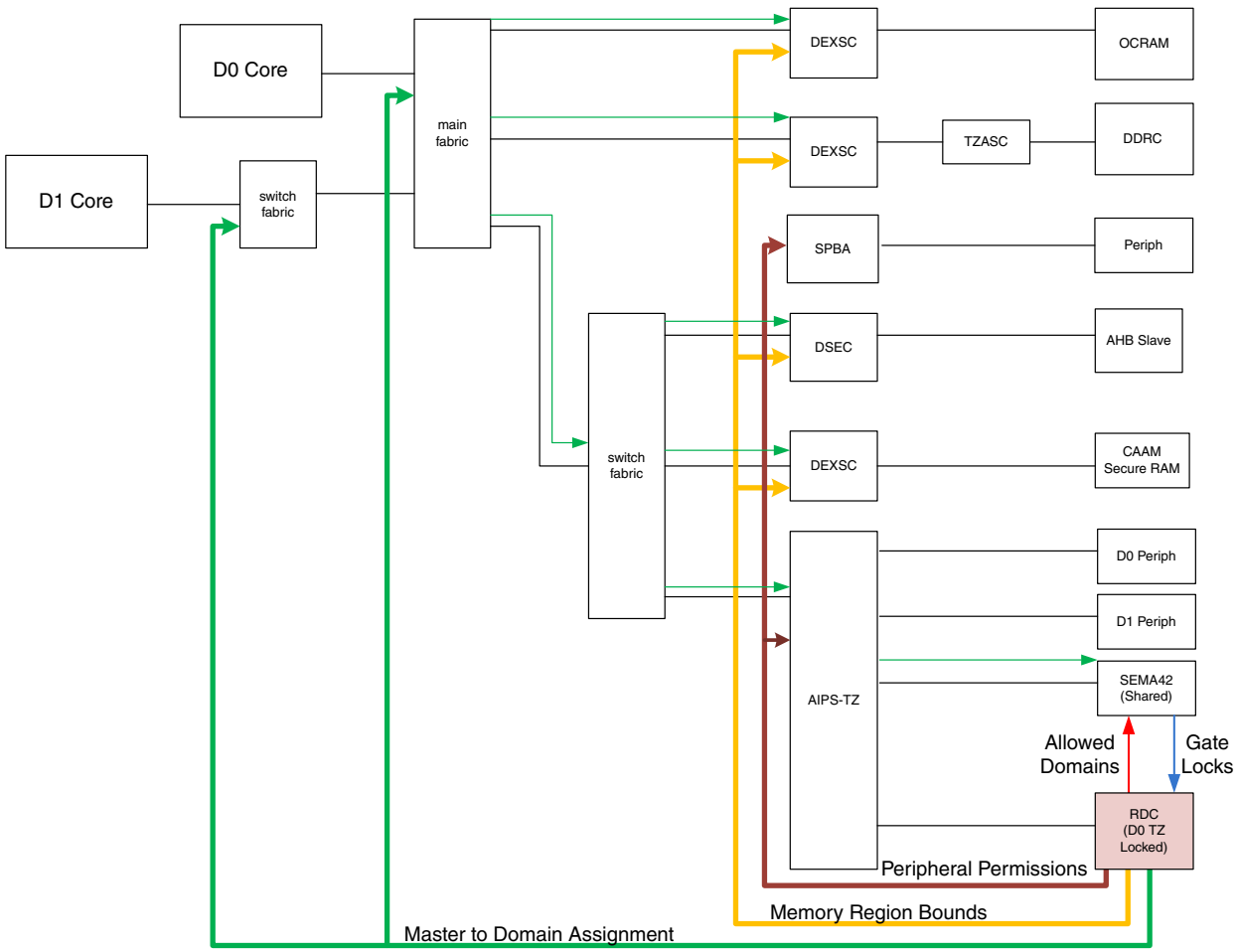


Figure 8-2. Example RDC Connections

For shared peripherals, RDC permits more than one domain access to a single peripheral. RDC also provides three ways to control synchronized use of shared peripherals. These methods include hardware-enforced synchronization, software-based semaphores, or no synchronization. The latter may be suitable for well-tuned multi-core operating systems that handle synchronization in the core platform, for instance.

For hardware-enforced synchronization, also known as "safe sharing", ownership of the peripheral must be claimed in the semaphore controller before access is allowed to the shared peripheral. The "semaphore required" bit (SREQ) is set in the PDAP register corresponding to the shared peripheral which causes the RDC to require that a semaphore is obtained by a domain before access by that domain to the shared peripheral is allowed. During the time that the domain has the semaphore in possession its bus masters have exclusive access to the peripheral.

When the semaphore is released then no domain masters have access until the semaphore is obtained again. When the SREQ is set, RDC does not allow masters to obtain semaphores of peripherals to which it is not allocated; the master must have designated access in the D-registers of the corresponding PDAP register (e.g. D3R bit set for Domain 3 access of the shared peripheral). There is a one-to-one mapping between the semaphore controller gate and the resource domain controller peripheral. The mapping of PDAP registers and peripherals can be found in the Peripheral Map section of the RDC chapter.

8.2.1 Domain ID

The RDC provides for an isolation of domain resources by use of a identifier called the Domain ID (DID). A core and its resources including memory, bus masters, and peripherals are all associated with a single DID. When software or a DMA attempts to access a peripheral or memory, the corresponding bus transaction includes the DID along with the other bus control information such as Read, Write, and privilege mode.

8.2.2 Resource Assignment

The RDC allows assignment of peripherals and memories to one or more domains while each bus master or core is placed in one of four domains. The masters are assigned a domain in the MDA register. A peripheral is given R/W access permissions to each domain in the PDAP register. Memory regions are bound by address space in start and end registers, the MRSA and MREA. Each memory region is assigned one or more allowed domains and R/W permissions in the MRC control register. Memory regions must be enabled before the permissions are active. Otherwise the permissions are not restricted.

The RDC itself should be isolated to ensure that only a trustworthy resource manager can configure the RDC registers. This process may either be present initially, during secure boot, or during the runtime in the secure world, for example. If the operating system does not support a runtime trusted execution then during the secure boot process the RDC configuration can be locked to prevent further modification after the operating systems are running.

NOTE

The CCM supports multicore awareness based on resource domain assignments programmed into the RDC. Refer to the CCM chapter regarding the relationship between core resource domains and their respective CCM resources. Failing to follow

the proper sequence when updating the resource domain assignments of the core can result in clocks being inadvertently gated.

8.2.3 Safe Sharing

For shared peripherals, the RDC can be configured to require a domain to obtain a semaphore lock before access to the peripheral is allowed. This feature helps prevent collisions from processes on separate cores that may want to use the same peripheral at the same time. The RDC sends a list of eligible domains to the semaphore module for each gate/peripheral. The eligible domains are those that are set in the peripheral domain access permissions (PDAP) registers. There is a one-to-one correspondence between semaphore gates and peripherals so each gate in the semaphore block represents a peripheral. The RDC receives semaphore locks from the hardware semaphore module (SEMA42). A semaphore lock is acquired when a core or bus master from a given domain requests a lock for a particular gate. The semaphore module compares the request's domain ID against the list of eligible domain IDs. If the domain ID is on the list and the lock is available then the lock is set and a signal is sent back to the RDC module indicating a lock has been acquired for a particular gate and to which domain ID the lock belongs. The RDC then restricts access to the corresponding peripheral to only transactions originating from the domain that has the lock. Another domain, though on the shared list to access the peripheral, must then wait until the lock is released before acquiring the lock and gaining access to the peripheral. To enable this feature of hardware enforcement for the semaphore locks, the SREQ bit is set in the RDC resource register.

If the SREQ is set, then when a process determines it needs a shared peripheral, it must first lock the resource in the semaphore module. Once the resource is locked, the semaphore module sends a signal to the RDC indicating the domain has access to the resource. The RDC will then set the access permissions to allow that domain access to the peripheral.

For a domain to acquire a lock on a peripheral, the domain must have been assigned to the peripheral in the RDC Peripheral Domain Access Permissions register (PDAP). The semaphore module only allows safe-sharing locks for those domains that are assigned to the peripheral. The semaphore module does not consider the access type (Read or Write) when allowing domains to acquire locks.

The SEMA42 module implements hardware-enforced semaphores as an IPS-mapped slave peripheral device. The feature set includes:

- Module definition supporting 64 hardware-enforced gates in a multi-processor configuration, where up to 15 processors can be supported; cp*X* is meant to represent core processor *X*
  - Gates appear as an *n*-entry byte-size array with read and write accesses (*n* = 16, 32, 64).
    - Processors lock gates by writing "Master_index" to the appropriate gate and must read back the gate value to verify the lock operation was successful. The Master_index value for the processors can be found in the Master Index Allocation table, which can be found in the AIPSTZ block.. Also note that after locking, the gate register contains the master_id value of the locking processor (in bits [3:0]), and also the value of the locking domain (in bits [5:4]).
    - Once locked, the gate is unlocked by a write of zeroes from the locking processor.
    - The number of implemented gates is specified by a hardware configuration define.
  - Each hardware gate appears as a 16-state, 4-bit state machine.
    - 16-state implementation

      if gate = 0x0, then state = unlocked
      if gate = 0x1, then state = locked by processor (master_index) 0
      if gate = 0x2, then state = locked by processor (master_index) 1
      …
      if gate = 0xF, then state = locked by processor (master_index) 14
    - Uses the logical bus master number (master_index) as a reference attribute plus the specified data patterns to validate all write operations.
    - Once locked, the gate can (and must) be unlocked by a write of zeroes from the locking processor.
  - Secure reset mechanisms are supported to clear the contents of individual gates, as well as a clear_all capability.
- Memory-mapped IPS slave peripheral platform module
  - Interface to the IPS bus for programming-model accesses

8.2.4 Resource Domain Control and Security Considerations

Conceptually, the RDC configuration is independent of the processor privilege mode and security domain. It is intended to allow for isolation between core processing environments to prevent collisions and increase reliability. Access between resource domains is mutually exclusive and each domain should be in control of its own privilege modes and access rights.

However, it is important to realize multi-core processors may have a multiple resource domains but only one overarching security domain. Chip security controls reside in one resource domain. In this configuration, a domain can affect at least one level of access privileges in the other domain. This may be acceptable but clarity and care is needed to ensure expected functionality.

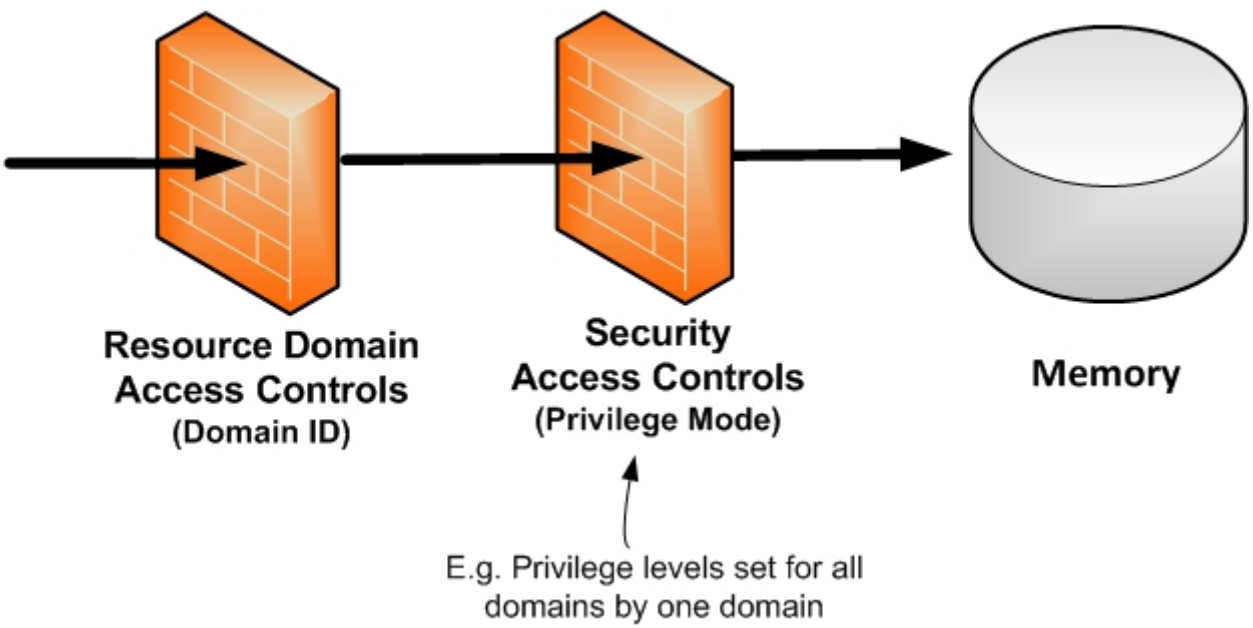


Figure 8-3. Access Control to Memory

Therefore, access to the security controls should be restricted to the most trustworthy operating mode of the core and privilege levels should be coordinated to ensure that shared peripherals and memory regions are accessible by both cores. For instance, if a memory region is designated for secure accesses then all domain masters that share that region must have secure privileges.

8.3 Modes of Operation

The RDC provides access controls to the resource domain subsystem. When the device is in a low power mode then some memory regions in the subsystem may be powered off. RDC responds to the impacted memory regions by automatically reconfiguring the memory regions once power returns and blocking access to those memory regions until the reconfiguration process is complete.

8.3.1 Low Power Modes

The RDC loads configuration information for memory regions (MRSA, MRSE, MRC) into access control mechanisms (gaskets) at the memory interface. The location of this configuration information may reside inside power domains that lose power during sleep modes for energy savings. To restore configuration information upon return from sleep mode, the RDC receives a global power control signal indicating power is restored. The RDC then automatically reconfigures the memory regions with the configuration information.

During reconfiguration, access is blocked to the previously powered down memories. When the RDC completes reconfiguration it issues an interrupt and allows access to the memory regions. Only the powered down regions are blocked during the reconfiguration. Memory regions in the "always-on" power domain (still powered during sleep mode) remain available according to the programmed access rights. If no memory regions were enabled then the powered down regions are available immediately when power is restored.

The figure below shows the Global Power Control signal which RDC uses to invalidate the configuration upon deassertion and to restore the configuration when re-asserted. The configuration is valid and bus transactions allowed once the memory regions have been restored.

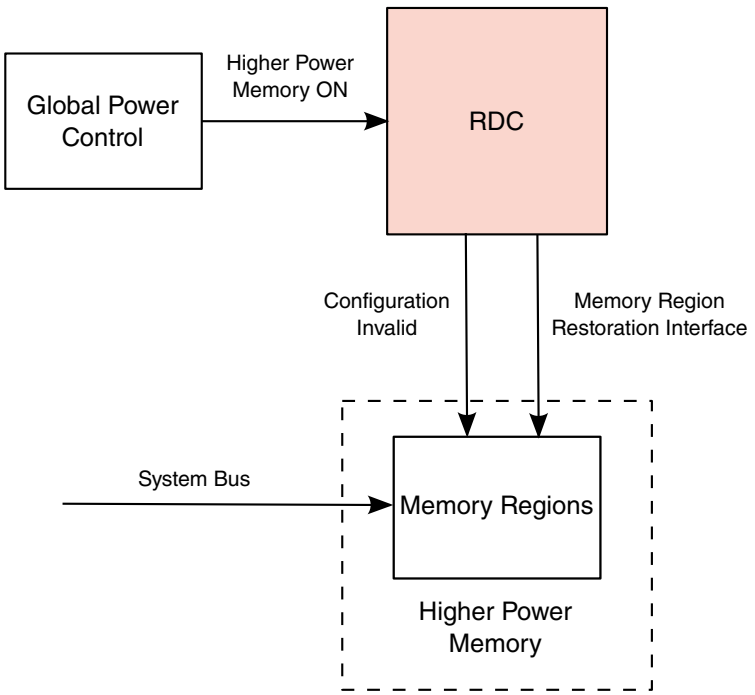


Figure 8-4. Memory Restoration Signaling

8.4 Programming Interface

This section provides product specific details describing the mapping of resources - peripherals, bus masters, and memory regions - to corresponding resource domain controls RDC registers.

The RDC and RDC_SEMA42 register maps are combined in this chapter. The base address for the one RDC map and two SEMA42 maps are each separated by 4KB. While there are two SEMA42 submodules and therefore two sets of SEMA42 registers, this chapter describes one. Please refer to the peripheral memory map for the base addresses of the RDC and SEMA42 modules.

8.4.1 Master Assignment Registers

Table 8-1. Master Assignment Mapping

Master	RDC MDA Register
Quad A53	RDC_MDA0

Table continues on the next page...

Table 8-1. Master Assignment Mapping (continued)

Master	RDC MDA Register
M7	RDC_MDA1
Reserved	RDC_MDA2
SDMA3 (p)	RDC_MDA3
Reserved	RDC_MDA4
LCDIF	RDC_MDA5
ISI	RDC_MDA6
SDMA3 (b)	RDC_MDA7
Coresight	RDC_MDA8
DAP	RDC_MDA9
CAAM	RDC_MDA10
SDMA1 (p)	RDC_MDA11
SDMA1 (b)	RDC_MDA12
APBHDMA	RDC_MDA13
NAND	RDC_MDA14
uSDHC1	RDC_MDA15
uSDHC2	RDC_MDA16
uSDHC3	RDC_MDA17
GPU	RDC_MDA18
USB1	RDC_MDA19
Reserved	RDC_MDA20
TESTPORT	RDC_MDA21
ENET1_TX	RDC_MDA22
ENET1_RX	RDC_MDA23
SDMA2 (p)	RDC_MDA24
SDMA2 (b)	RDC_MDA24
SDMA2 to SPBA2	RDC_MDA24
SDMA3 to SPBA2	RDC_MDA25
SDMA1 to SPBA1	RDC_MDA26

8.4.2 Peripheral Mapping

Each peripheral has a corresponding resource domain assignment register in the RDC and semaphore lock register in the RDC_SEMA42 module. The following table shows allocation of the RDC PDAP and RDC_SEMA4 GATE registers for peripheral resource domain assignment.

NOTE

Access control of the RDC registers can be programmed using the respective PDAP register. The default setting of the PDAP register for the RDC allows access from all domains. Use caution when restricting access of the RDC registers to avoid conditions where access to the RDC registers is needed but no master is assigned to a domain with access rights to the RDC.

Table 8-2. RDC Peripheral Mapping

Peripheral	RDC PDAP register	RDC_SEMA42 block/gate register
GPIO1	RDC_PDAP00	SEMA42 B1 / G0
GPIO2	RDC_PDAP01	SEMA42 B1 / G1
GPIO3	RDC_PDAP02	SEMA42 B1 / G2
GPIO4	RDC_PDAP03	SEMA42 B1 / G3
GPIO5	RDC_PDAP04	SEMA42 B1 / G4
Reserved	RDC_PDAP05	SEMA42 B1 / G5
ANA_TSENSOR	RDC_PDAP06	SEMA42 B1 / G6
ANA_OSC	RDC_PDAP07	SEMA42 B1 / G7
WDOG1	RDC_PDAP08	SEMA42 B1 / G8
WDOG2	RDC_PDAP09	SEMA42 B1 / G9
WDOG3	RDC_PDAP10	SEMA42 B1 / G10
SDMA3	RDC_PDAP11	SEMA42 B1 / G11
SDMA2	RDC_PDAP12	SEMA42 B1 / G12
GPT1	RDC_PDAP13	SEMA42 B1 / G13
GPT2	RDC_PDAP14	SEMA42 B1 / G14
GPT3	RDC_PDAP15	SEMA42 B1 / G15
Reserved	RDC_PDAP16	SEMA42 B1 / G16
ROMCP	RDC_PDAP17	SEMA42 B1 / G17
Reserved	RDC_PDAP18	SEMA42 B1 / G18
IOMUXC	RDC_PDAP19	SEMA42 B1 / G19
IOMUXC_GPR	RDC_PDAP20	SEMA42 B1 / G20
OCOTP_CTRL	RDC_PDAP21	SEMA42 B1 / G21
ANA_PLL	RDC_PDAP22	SEMA42 B1 / G22
SNVS_HP	RDC_PDAP23	SEMA42 B1 / G23
CCM	RDC_PDAP24	SEMA42 B1 / G24
SRC	RDC_PDAP25	SEMA42 B1 / G25
GPC	RDC_PDAP26	SEMA42 B1 / G26
SEMAPHORE1	RDC_PDAP27	SEMA42 B1 / G27
SEMAPHORE2	RDC_PDAP28	SEMA42 B1 / G28
RDC	RDC_PDAP29	SEMA42 B1 / G29

Table continues on the next page...

Table 8-2. RDC Peripheral Mapping (continued)

Peripheral	RDC PDAP register	RDC_SEMA42 block/gate register
CSU	RDC_PDAP30	SEMA42 B1 / G30
Reserved	RDC_PDAP31	SEMA42 B1 / G31
LCDIF	RDC_PDAP32	SEMA42 B1 / G32
MIPI_DSI	RDC_PDAP33	SEMA42 B1 / G33
ISI	RDC_PDAP34	SEMA42 B1 / G34
MIPI_CSI	RDC_PDAP35	SEMA42 B1 / G35
USB1	RDC_PDAP36	SEMA42 B1 / G36
Reserved	RDC_PDAP37	SEMA42 B1 / G37
PWM1	RDC_PDAP38	SEMA42 B1 / G38
PWM2	RDC_PDAP39	SEMA42 B1 / G39
PWM3	RDC_PDAP40	SEMA42 B1 / G40
PWM4	RDC_PDAP41	SEMA42 B1 / G41
System_Counter_RD	RDC_PDAP42	SEMA42 B1 / G42
System_Counter_CMP	RDC_PDAP43	SEMA42 B1 / G43
System_Counter_CTRL	RDC_PDAP44	SEMA42 B1 / G44
Reserved	RDC_PDAP45	SEMA42 B1 / G45
GPT6	RDC_PDAP46	SEMA42 B1 / G46
GPT5	RDC_PDAP47	SEMA42 B1 / G47
GPT4	RDC_PDAP48	SEMA42 B1 / G48
Reserved	RDC_PDAP49	SEMA42 B1 / G49
Reserved	RDC_PDAP50	SEMA42 B1 / G50
Reserved	RDC_PDAP51	SEMA42 B1 / G51
Reserved	RDC_PDAP52	SEMA42 B1 / G52
Reserved	RDC_PDAP53	SEMA42 B1 / G53
Reserved	RDC_PDAP54	SEMA42 B1 / G54
Reserved	RDC_PDAP55	SEMA42 B1 / G55
TZASC	RDC_PDAP56	SEMA42 B1 / G56
Reserved	RDC_PDAP57	SEMA42 B1 / G57
Reserved	RDC_PDAP58	SEMA42 B1 / G58
Reserved	RDC_PDAP59	SEMA42 B1 / G59
PERFMON1	RDC_PDAP60	SEMA42 B1 / G60
PERFMON2	RDC_PDAP61	SEMA42 B1 / G61
PLATFORM_CTRL	RDC_PDAP62	SEMA42 B1 / G62
QoSC	RDC_PDAP63	SEMA42 B1 / G63
Reserved	RDC_PDAP64	SEMA42 B2 / G0
Reserved	RDC_PDAP65	SEMA42 B2 / G1
I2C1	RDC_PDAP66	SEMA42 B2 / G2
I2C2	RDC_PDAP67	SEMA42 B2 / G3
I2C3	RDC_PDAP68	SEMA42 B2 / G4

Table continues on the next page...

Table 8-2. RDC Peripheral Mapping (continued)

Peripheral	RDC PDAP register	RDC_SEMA42 block/gate register
I2C4	RDC_PDAP69	SEMA42 B2 / G5
UART4	RDC_PDAP70	SEMA42 B2 / G6
Reserved	RDC_PDAP71	SEMA42 B2 / G7
Reserved	RDC_PDAP72	SEMA42 B2 / G8
Reserved	RDC_PDAP73	SEMA42 B2 / G9
MU_A	RDC_PDAP74	SEMA42 B2 / G10
MU_B	RDC_PDAP75	SEMA42 B2 / G11
SEMAPHORE_HS	RDC_PDAP76	SEMA42 B2 / G12
Reserved for SDMA2 internal memory	RDC_PDAP77	SEMA42 B2 / G13
Reserved	RDC_PDAP78	SEMA42 B2 / G14
SAI2	RDC_PDAP79	SEMA42 B2 / G15
SAI3	RDC_PDAP80	SEMA42 B2 / G16
Reserved	RDC_PDAP81	SEMA42 B2 / G17
SAI5	RDC_PDAP82	SEMA42 B2 / G18
SAI6	RDC_PDAP83	SEMA42 B2 / G19
uSDHC1	RDC_PDAP84	SEMA42 B2 / G20
uSDHC2	RDC_PDAP85	SEMA42 B2 / G21
uSDHC3	RDC_PDAP86	SEMA42 B2 / G22
SAI7	RDC_PDAP87	SEMA42 B2 / G23
Reserved	RDC_PDAP88	SEMA42 B2 / G24
Reserved for SDMA2 internal memory	RDC_PDAP89	SEMA42 B2 / G25
SPBA2	RDC_PDAP90	SEMA42 B2 / G26
QSPI	RDC_PDAP91	SEMA42 B2 / G27
Reserved	RDC_PDAP92	SEMA42 B2 / G28
SDMA1	RDC_PDAP93	SEMA42 B2 / G29
ENET1	RDC_PDAP94	SEMA42 B2 / G30
Reserved	RDC_PDAP95	SEMA42 B2 / G31
Reserved for SDMA internal memory	RDC_PDAP96	SEMA42 B2 / G32
SPDIF1	RDC_PDAP97	SEMA42 B2 / G33
eCSPI1	RDC_PDAP98	SEMA42 B2 / G34
eCSPI2	RDC_PDAP99	SEMA42 B2 / G35
eCSPI3	RDC_PDAP100	SEMA42 B2 / G36
MICFIL	RDC_PDAP101	SEMA42 B2 / G37
UART1	RDC_PDAP102	SEMA42 B2 / G38
Reserved for SDMA internal registers	RDC_PDAP103	SEMA42 B2 / G39
UART3	RDC_PDAP104	SEMA42 B2 / G40
UART2	RDC_PDAP105	SEMA42 B2 / G41
Reserved	RDC_PDAP106	SEMA42 B2 / G42
ASRC	RDC_PDAP107	SEMA42 B2 / G43

Table continues on the next page...

Table 8-2. RDC Peripheral Mapping (continued)

Peripheral	RDC PDAP register	RDC_SEMA42 block/gate register
Reserved	RDC_PDAP108	SEMA42 B2 / G44
Reserved	RDC_PDAP109	SEMA42 B2 / G45
Reserved for SDMA internal registers	RDC_PDAP110	SEMA42 B2 / G46
SPBA1	RDC_PDAP111	SEMA42 B2 / G47
module_en_glbl[0]	RDC_PDAP112	SEMA42 B2 / G48
module_en_glbl[0]	RDC_PDAP113	SEMA42 B2 / G49
CAAM	RDC_PDAP114	SEMA42 B2 / G50

8.4.3 Memory Region Map

The number of memories with domain isolation support varies per device. The number of memory regions for a particular memory and the size of those regions varies per memory gasket. Each region of memory has a set of registers to define the boundaries of the region based on start and end addresses, a control register to set the domain access permissions and enable the region, and a status register to determin if access was denied to a region.

For this device, refer to the table below to determine the memories with domain support, the number of regions for each memory, the region resolution, the identifying numbers for the sets of memory region registers, and the addresses of the RDC registers to access the sets of Memory Region registers.

8.5 RDC Memory Map/Register Definition

RDC memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0000	Version Information (RDC_VIR)	32	R	0376_E204h	8.5.1/245
303D_0024	Status (RDC_STAT)	32	R/W	0000_0100h	8.5.2/246
303D_0028	Interrupt and Control (RDC_INTCTRL)	32	R/W	0000_0000h	8.5.3/247
303D_002C	Interrupt Status (RDC_INTSTAT)	32	R/W	See section	8.5.4/247
303D_0200	Master Domain Assignment (RDC_MDA0)	32	R/W	0000_0000h	8.5.5/248
303D_0204	Master Domain Assignment (RDC_MDA1)	32	R/W	0000_0000h	8.5.5/248
303D_0208	Master Domain Assignment (RDC_MDA2)	32	R/W	0000_0000h	8.5.5/248

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_020C	Master Domain Assignment (RDC_MDA3)	32	R/W	0000_0000h	8.5.5/248
303D_0210	Master Domain Assignment (RDC_MDA4)	32	R/W	0000_0000h	8.5.5/248
303D_0214	Master Domain Assignment (RDC_MDA5)	32	R/W	0000_0000h	8.5.5/248
303D_0218	Master Domain Assignment (RDC_MDA6)	32	R/W	0000_0000h	8.5.5/248
303D_021C	Master Domain Assignment (RDC_MDA7)	32	R/W	0000_0000h	8.5.5/248
303D_0220	Master Domain Assignment (RDC_MDA8)	32	R/W	0000_0000h	8.5.5/248
303D_0224	Master Domain Assignment (RDC_MDA9)	32	R/W	0000_0000h	8.5.5/248
303D_0228	Master Domain Assignment (RDC_MDA10)	32	R/W	0000_0000h	8.5.5/248
303D_022C	Master Domain Assignment (RDC_MDA11)	32	R/W	0000_0000h	8.5.5/248
303D_0230	Master Domain Assignment (RDC_MDA12)	32	R/W	0000_0000h	8.5.5/248
303D_0234	Master Domain Assignment (RDC_MDA13)	32	R/W	0000_0000h	8.5.5/248
303D_0238	Master Domain Assignment (RDC_MDA14)	32	R/W	0000_0000h	8.5.5/248
303D_023C	Master Domain Assignment (RDC_MDA15)	32	R/W	0000_0000h	8.5.5/248
303D_0240	Master Domain Assignment (RDC_MDA16)	32	R/W	0000_0000h	8.5.5/248
303D_0244	Master Domain Assignment (RDC_MDA17)	32	R/W	0000_0000h	8.5.5/248
303D_0248	Master Domain Assignment (RDC_MDA18)	32	R/W	0000_0000h	8.5.5/248
303D_024C	Master Domain Assignment (RDC_MDA19)	32	R/W	0000_0000h	8.5.5/248
303D_0250	Master Domain Assignment (RDC_MDA20)	32	R/W	0000_0000h	8.5.5/248
303D_0254	Master Domain Assignment (RDC_MDA21)	32	R/W	0000_0000h	8.5.5/248
303D_0258	Master Domain Assignment (RDC_MDA22)	32	R/W	0000_0000h	8.5.5/248
303D_025C	Master Domain Assignment (RDC_MDA23)	32	R/W	0000_0000h	8.5.5/248
303D_0260	Master Domain Assignment (RDC_MDA24)	32	R/W	0000_0000h	8.5.5/248
303D_0264	Master Domain Assignment (RDC_MDA25)	32	R/W	0000_0000h	8.5.5/248
303D_0268	Master Domain Assignment (RDC_MDA26)	32	R/W	0000_0000h	8.5.5/248
303D_0400	Peripheral Domain Access Permissions (RDC_PDAP0)	32	R/W	0000_00FFh	8.5.6/249
303D_0404	Peripheral Domain Access Permissions (RDC_PDAP1)	32	R/W	0000_00FFh	8.5.6/249
303D_0408	Peripheral Domain Access Permissions (RDC_PDAP2)	32	R/W	0000_00FFh	8.5.6/249
303D_040C	Peripheral Domain Access Permissions (RDC_PDAP3)	32	R/W	0000_00FFh	8.5.6/249
303D_0410	Peripheral Domain Access Permissions (RDC_PDAP4)	32	R/W	0000_00FFh	8.5.6/249
303D_0414	Peripheral Domain Access Permissions (RDC_PDAP5)	32	R/W	0000_00FFh	8.5.6/249
303D_0418	Peripheral Domain Access Permissions (RDC_PDAP6)	32	R/W	0000_00FFh	8.5.6/249
303D_041C	Peripheral Domain Access Permissions (RDC_PDAP7)	32	R/W	0000_00FFh	8.5.6/249
303D_0420	Peripheral Domain Access Permissions (RDC_PDAP8)	32	R/W	0000_00FFh	8.5.6/249
303D_0424	Peripheral Domain Access Permissions (RDC_PDAP9)	32	R/W	0000_00FFh	8.5.6/249
303D_0428	Peripheral Domain Access Permissions (RDC_PDAP10)	32	R/W	0000_00FFh	8.5.6/249
303D_042C	Peripheral Domain Access Permissions (RDC_PDAP11)	32	R/W	0000_00FFh	8.5.6/249
303D_0430	Peripheral Domain Access Permissions (RDC_PDAP12)	32	R/W	0000_00FFh	8.5.6/249
303D_0434	Peripheral Domain Access Permissions (RDC_PDAP13)	32	R/W	0000_00FFh	8.5.6/249

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0438	Peripheral Domain Access Permissions (RDC_PDAP14)	32	R/W	0000_00FFh	8.5.6/249
303D_043C	Peripheral Domain Access Permissions (RDC_PDAP15)	32	R/W	0000_00FFh	8.5.6/249
303D_0440	Peripheral Domain Access Permissions (RDC_PDAP16)	32	R/W	0000_00FFh	8.5.6/249
303D_0444	Peripheral Domain Access Permissions (RDC_PDAP17)	32	R/W	0000_00FFh	8.5.6/249
303D_0448	Peripheral Domain Access Permissions (RDC_PDAP18)	32	R/W	0000_00FFh	8.5.6/249
303D_044C	Peripheral Domain Access Permissions (RDC_PDAP19)	32	R/W	0000_00FFh	8.5.6/249
303D_0450	Peripheral Domain Access Permissions (RDC_PDAP20)	32	R/W	0000_00FFh	8.5.6/249
303D_0454	Peripheral Domain Access Permissions (RDC_PDAP21)	32	R/W	0000_00FFh	8.5.6/249
303D_0458	Peripheral Domain Access Permissions (RDC_PDAP22)	32	R/W	0000_00FFh	8.5.6/249
303D_045C	Peripheral Domain Access Permissions (RDC_PDAP23)	32	R/W	0000_00FFh	8.5.6/249
303D_0460	Peripheral Domain Access Permissions (RDC_PDAP24)	32	R/W	0000_00FFh	8.5.6/249
303D_0464	Peripheral Domain Access Permissions (RDC_PDAP25)	32	R/W	0000_00FFh	8.5.6/249
303D_0468	Peripheral Domain Access Permissions (RDC_PDAP26)	32	R/W	0000_00FFh	8.5.6/249
303D_046C	Peripheral Domain Access Permissions (RDC_PDAP27)	32	R/W	0000_00FFh	8.5.6/249
303D_0470	Peripheral Domain Access Permissions (RDC_PDAP28)	32	R/W	0000_00FFh	8.5.6/249
303D_0474	Peripheral Domain Access Permissions (RDC_PDAP29)	32	R/W	0000_00FFh	8.5.6/249
303D_0478	Peripheral Domain Access Permissions (RDC_PDAP30)	32	R/W	0000_00FFh	8.5.6/249
303D_047C	Peripheral Domain Access Permissions (RDC_PDAP31)	32	R/W	0000_00FFh	8.5.6/249
303D_0480	Peripheral Domain Access Permissions (RDC_PDAP32)	32	R/W	0000_00FFh	8.5.6/249
303D_0484	Peripheral Domain Access Permissions (RDC_PDAP33)	32	R/W	0000_00FFh	8.5.6/249
303D_0488	Peripheral Domain Access Permissions (RDC_PDAP34)	32	R/W	0000_00FFh	8.5.6/249
303D_048C	Peripheral Domain Access Permissions (RDC_PDAP35)	32	R/W	0000_00FFh	8.5.6/249
303D_0490	Peripheral Domain Access Permissions (RDC_PDAP36)	32	R/W	0000_00FFh	8.5.6/249
303D_0494	Peripheral Domain Access Permissions (RDC_PDAP37)	32	R/W	0000_00FFh	8.5.6/249
303D_0498	Peripheral Domain Access Permissions (RDC_PDAP38)	32	R/W	0000_00FFh	8.5.6/249
303D_049C	Peripheral Domain Access Permissions (RDC_PDAP39)	32	R/W	0000_00FFh	8.5.6/249
303D_04A0	Peripheral Domain Access Permissions (RDC_PDAP40)	32	R/W	0000_00FFh	8.5.6/249
303D_04A4	Peripheral Domain Access Permissions (RDC_PDAP41)	32	R/W	0000_00FFh	8.5.6/249
303D_04A8	Peripheral Domain Access Permissions (RDC_PDAP42)	32	R/W	0000_00FFh	8.5.6/249
303D_04AC	Peripheral Domain Access Permissions (RDC_PDAP43)	32	R/W	0000_00FFh	8.5.6/249
303D_04B0	Peripheral Domain Access Permissions (RDC_PDAP44)	32	R/W	0000_00FFh	8.5.6/249
303D_04B4	Peripheral Domain Access Permissions (RDC_PDAP45)	32	R/W	0000_00FFh	8.5.6/249
303D_04B8	Peripheral Domain Access Permissions (RDC_PDAP46)	32	R/W	0000_00FFh	8.5.6/249
303D_04BC	Peripheral Domain Access Permissions (RDC_PDAP47)	32	R/W	0000_00FFh	8.5.6/249
303D_04C0	Peripheral Domain Access Permissions (RDC_PDAP48)	32	R/W	0000_00FFh	8.5.6/249
303D_04C4	Peripheral Domain Access Permissions (RDC_PDAP49)	32	R/W	0000_00FFh	8.5.6/249
303D_04C8	Peripheral Domain Access Permissions (RDC_PDAP50)	32	R/W	0000_00FFh	8.5.6/249
303D_04CC	Peripheral Domain Access Permissions (RDC_PDAP51)	32	R/W	0000_00FFh	8.5.6/249

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_04D0	Peripheral Domain Access Permissions (RDC_PDAP52)	32	R/W	0000_00FFh	8.5.6/249
303D_04D4	Peripheral Domain Access Permissions (RDC_PDAP53)	32	R/W	0000_00FFh	8.5.6/249
303D_04D8	Peripheral Domain Access Permissions (RDC_PDAP54)	32	R/W	0000_00FFh	8.5.6/249
303D_04DC	Peripheral Domain Access Permissions (RDC_PDAP55)	32	R/W	0000_00FFh	8.5.6/249
303D_04E0	Peripheral Domain Access Permissions (RDC_PDAP56)	32	R/W	0000_00FFh	8.5.6/249
303D_04E4	Peripheral Domain Access Permissions (RDC_PDAP57)	32	R/W	0000_00FFh	8.5.6/249
303D_04E8	Peripheral Domain Access Permissions (RDC_PDAP58)	32	R/W	0000_00FFh	8.5.6/249
303D_04EC	Peripheral Domain Access Permissions (RDC_PDAP59)	32	R/W	0000_00FFh	8.5.6/249
303D_04F0	Peripheral Domain Access Permissions (RDC_PDAP60)	32	R/W	0000_00FFh	8.5.6/249
303D_04F4	Peripheral Domain Access Permissions (RDC_PDAP61)	32	R/W	0000_00FFh	8.5.6/249
303D_04F8	Peripheral Domain Access Permissions (RDC_PDAP62)	32	R/W	0000_00FFh	8.5.6/249
303D_04FC	Peripheral Domain Access Permissions (RDC_PDAP63)	32	R/W	0000_00FFh	8.5.6/249
303D_0500	Peripheral Domain Access Permissions (RDC_PDAP64)	32	R/W	0000_00FFh	8.5.6/249
303D_0504	Peripheral Domain Access Permissions (RDC_PDAP65)	32	R/W	0000_00FFh	8.5.6/249
303D_0508	Peripheral Domain Access Permissions (RDC_PDAP66)	32	R/W	0000_00FFh	8.5.6/249
303D_050C	Peripheral Domain Access Permissions (RDC_PDAP67)	32	R/W	0000_00FFh	8.5.6/249
303D_0510	Peripheral Domain Access Permissions (RDC_PDAP68)	32	R/W	0000_00FFh	8.5.6/249
303D_0514	Peripheral Domain Access Permissions (RDC_PDAP69)	32	R/W	0000_00FFh	8.5.6/249
303D_0518	Peripheral Domain Access Permissions (RDC_PDAP70)	32	R/W	0000_00FFh	8.5.6/249
303D_051C	Peripheral Domain Access Permissions (RDC_PDAP71)	32	R/W	0000_00FFh	8.5.6/249
303D_0520	Peripheral Domain Access Permissions (RDC_PDAP72)	32	R/W	0000_00FFh	8.5.6/249
303D_0524	Peripheral Domain Access Permissions (RDC_PDAP73)	32	R/W	0000_00FFh	8.5.6/249
303D_0528	Peripheral Domain Access Permissions (RDC_PDAP74)	32	R/W	0000_00FFh	8.5.6/249
303D_052C	Peripheral Domain Access Permissions (RDC_PDAP75)	32	R/W	0000_00FFh	8.5.6/249
303D_0530	Peripheral Domain Access Permissions (RDC_PDAP76)	32	R/W	0000_00FFh	8.5.6/249
303D_0534	Peripheral Domain Access Permissions (RDC_PDAP77)	32	R/W	0000_00FFh	8.5.6/249
303D_0538	Peripheral Domain Access Permissions (RDC_PDAP78)	32	R/W	0000_00FFh	8.5.6/249
303D_053C	Peripheral Domain Access Permissions (RDC_PDAP79)	32	R/W	0000_00FFh	8.5.6/249
303D_0540	Peripheral Domain Access Permissions (RDC_PDAP80)	32	R/W	0000_00FFh	8.5.6/249
303D_0544	Peripheral Domain Access Permissions (RDC_PDAP81)	32	R/W	0000_00FFh	8.5.6/249
303D_0548	Peripheral Domain Access Permissions (RDC_PDAP82)	32	R/W	0000_00FFh	8.5.6/249
303D_054C	Peripheral Domain Access Permissions (RDC_PDAP83)	32	R/W	0000_00FFh	8.5.6/249
303D_0550	Peripheral Domain Access Permissions (RDC_PDAP84)	32	R/W	0000_00FFh	8.5.6/249
303D_0554	Peripheral Domain Access Permissions (RDC_PDAP85)	32	R/W	0000_00FFh	8.5.6/249
303D_0558	Peripheral Domain Access Permissions (RDC_PDAP86)	32	R/W	0000_00FFh	8.5.6/249
303D_055C	Peripheral Domain Access Permissions (RDC_PDAP87)	32	R/W	0000_00FFh	8.5.6/249
303D_0560	Peripheral Domain Access Permissions (RDC_PDAP88)	32	R/W	0000_00FFh	8.5.6/249
303D_0564	Peripheral Domain Access Permissions (RDC_PDAP89)	32	R/W	0000_00FFh	8.5.6/249

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0568	Peripheral Domain Access Permissions (RDC_PDAP90)	32	R/W	0000_00FFh	8.5.6/249
303D_056C	Peripheral Domain Access Permissions (RDC_PDAP91)	32	R/W	0000_00FFh	8.5.6/249
303D_0570	Peripheral Domain Access Permissions (RDC_PDAP92)	32	R/W	0000_00FFh	8.5.6/249
303D_0574	Peripheral Domain Access Permissions (RDC_PDAP93)	32	R/W	0000_00FFh	8.5.6/249
303D_0578	Peripheral Domain Access Permissions (RDC_PDAP94)	32	R/W	0000_00FFh	8.5.6/249
303D_057C	Peripheral Domain Access Permissions (RDC_PDAP95)	32	R/W	0000_00FFh	8.5.6/249
303D_0580	Peripheral Domain Access Permissions (RDC_PDAP96)	32	R/W	0000_00FFh	8.5.6/249
303D_0584	Peripheral Domain Access Permissions (RDC_PDAP97)	32	R/W	0000_00FFh	8.5.6/249
303D_0588	Peripheral Domain Access Permissions (RDC_PDAP98)	32	R/W	0000_00FFh	8.5.6/249
303D_058C	Peripheral Domain Access Permissions (RDC_PDAP99)	32	R/W	0000_00FFh	8.5.6/249
303D_0590	Peripheral Domain Access Permissions (RDC_PDAP100)	32	R/W	0000_00FFh	8.5.6/249
303D_0594	Peripheral Domain Access Permissions (RDC_PDAP101)	32	R/W	0000_00FFh	8.5.6/249
303D_0598	Peripheral Domain Access Permissions (RDC_PDAP102)	32	R/W	0000_00FFh	8.5.6/249
303D_059C	Peripheral Domain Access Permissions (RDC_PDAP103)	32	R/W	0000_00FFh	8.5.6/249
303D_05A0	Peripheral Domain Access Permissions (RDC_PDAP104)	32	R/W	0000_00FFh	8.5.6/249
303D_05A4	Peripheral Domain Access Permissions (RDC_PDAP105)	32	R/W	0000_00FFh	8.5.6/249
303D_05A8	Peripheral Domain Access Permissions (RDC_PDAP106)	32	R/W	0000_00FFh	8.5.6/249
303D_05AC	Peripheral Domain Access Permissions (RDC_PDAP107)	32	R/W	0000_00FFh	8.5.6/249
303D_05B0	Peripheral Domain Access Permissions (RDC_PDAP108)	32	R/W	0000_00FFh	8.5.6/249
303D_05B4	Peripheral Domain Access Permissions (RDC_PDAP109)	32	R/W	0000_00FFh	8.5.6/249
303D_05B8	Peripheral Domain Access Permissions (RDC_PDAP110)	32	R/W	0000_00FFh	8.5.6/249
303D_05BC	Peripheral Domain Access Permissions (RDC_PDAP111)	32	R/W	0000_00FFh	8.5.6/249
303D_05C0	Peripheral Domain Access Permissions (RDC_PDAP112)	32	R/W	0000_00FFh	8.5.6/249
303D_05C4	Peripheral Domain Access Permissions (RDC_PDAP113)	32	R/W	0000_00FFh	8.5.6/249
303D_05C8	Peripheral Domain Access Permissions (RDC_PDAP114)	32	R/W	0000_00FFh	8.5.6/249
303D_05CC	Peripheral Domain Access Permissions (RDC_PDAP115)	32	R/W	0000_00FFh	8.5.6/249
303D_05D0	Peripheral Domain Access Permissions (RDC_PDAP116)	32	R/W	0000_00FFh	8.5.6/249
303D_05D4	Peripheral Domain Access Permissions (RDC_PDAP117)	32	R/W	0000_00FFh	8.5.6/249
303D_0800	Memory Region Start Address (RDC_MRSA0)	32	R/W	Undefined	8.5.7/250
303D_0804	Memory Region End Address (RDC_MREA0)	32	R/W	Undefined	8.5.8/252
303D_0808	Memory Region Control (RDC_MRC0)	32	R/W	0000_00FFh	8.5.9/253
303D_080C	Memory Region Violation Status (RDC_MRVS0)	32	R/W	0000_0000h	8.5.10/254
303D_0810	Memory Region Start Address (RDC_MRSA1)	32	R/W	Undefined	8.5.7/250
303D_0814	Memory Region End Address (RDC_MREA1)	32	R/W	Undefined	8.5.8/252
303D_0818	Memory Region Control (RDC_MRC1)	32	R/W	0000_00FFh	8.5.9/253
303D_081C	Memory Region Violation Status (RDC_MRVS1)	32	R/W	0000_0000h	8.5.10/254
303D_0820	Memory Region Start Address (RDC_MRSA2)	32	R/W	Undefined	8.5.7/250
303D_0824	Memory Region End Address (RDC_MREA2)	32	R/W	Undefined	8.5.8/252

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0828	Memory Region Control (RDC_MRC2)	32	R/W	0000_00FFh	8.5.9/253
303D_082C	Memory Region Violation Status (RDC_MRVS2)	32	R/W	0000_0000h	8.5.10/254
303D_0830	Memory Region Start Address (RDC_MRSA3)	32	R/W	Undefined	8.5.7/250
303D_0834	Memory Region End Address (RDC_MREA3)	32	R/W	Undefined	8.5.8/252
303D_0838	Memory Region Control (RDC_MRC3)	32	R/W	0000_00FFh	8.5.9/253
303D_083C	Memory Region Violation Status (RDC_MRVS3)	32	R/W	0000_0000h	8.5.10/254
303D_0840	Memory Region Start Address (RDC_MRSA4)	32	R/W	Undefined	8.5.7/250
303D_0844	Memory Region End Address (RDC_MREA4)	32	R/W	Undefined	8.5.8/252
303D_0848	Memory Region Control (RDC_MRC4)	32	R/W	0000_00FFh	8.5.9/253
303D_084C	Memory Region Violation Status (RDC_MRVS4)	32	R/W	0000_0000h	8.5.10/254
303D_0850	Memory Region Start Address (RDC_MRSA5)	32	R/W	Undefined	8.5.7/250
303D_0854	Memory Region End Address (RDC_MREA5)	32	R/W	Undefined	8.5.8/252
303D_0858	Memory Region Control (RDC_MRC5)	32	R/W	0000_00FFh	8.5.9/253
303D_085C	Memory Region Violation Status (RDC_MRVS5)	32	R/W	0000_0000h	8.5.10/254
303D_0860	Memory Region Start Address (RDC_MRSA6)	32	R/W	Undefined	8.5.7/250
303D_0864	Memory Region End Address (RDC_MREA6)	32	R/W	Undefined	8.5.8/252
303D_0868	Memory Region Control (RDC_MRC6)	32	R/W	0000_00FFh	8.5.9/253
303D_086C	Memory Region Violation Status (RDC_MRVS6)	32	R/W	0000_0000h	8.5.10/254
303D_0870	Memory Region Start Address (RDC_MRSA7)	32	R/W	Undefined	8.5.7/250
303D_0874	Memory Region End Address (RDC_MREA7)	32	R/W	Undefined	8.5.8/252
303D_0878	Memory Region Control (RDC_MRC7)	32	R/W	0000_00FFh	8.5.9/253
303D_087C	Memory Region Violation Status (RDC_MRVS7)	32	R/W	0000_0000h	8.5.10/254
303D_0880	Memory Region Start Address (RDC_MRSA8)	32	R/W	Undefined	8.5.7/250
303D_0884	Memory Region End Address (RDC_MREA8)	32	R/W	Undefined	8.5.8/252
303D_0888	Memory Region Control (RDC_MRC8)	32	R/W	0000_00FFh	8.5.9/253
303D_088C	Memory Region Violation Status (RDC_MRVS8)	32	R/W	0000_0000h	8.5.10/254
303D_0890	Memory Region Start Address (RDC_MRSA9)	32	R/W	Undefined	8.5.7/250
303D_0894	Memory Region End Address (RDC_MREA9)	32	R/W	Undefined	8.5.8/252
303D_0898	Memory Region Control (RDC_MRC9)	32	R/W	0000_00FFh	8.5.9/253
303D_089C	Memory Region Violation Status (RDC_MRVS9)	32	R/W	0000_0000h	8.5.10/254
303D_08A0	Memory Region Start Address (RDC_MRSA10)	32	R/W	Undefined	8.5.7/250
303D_08A4	Memory Region End Address (RDC_MREA10)	32	R/W	Undefined	8.5.8/252
303D_08A8	Memory Region Control (RDC_MRC10)	32	R/W	0000_00FFh	8.5.9/253
303D_08AC	Memory Region Violation Status (RDC_MRVS10)	32	R/W	0000_0000h	8.5.10/254
303D_08B0	Memory Region Start Address (RDC_MRSA11)	32	R/W	Undefined	8.5.7/250
303D_08B4	Memory Region End Address (RDC_MREA11)	32	R/W	Undefined	8.5.8/252
303D_08B8	Memory Region Control (RDC_MRC11)	32	R/W	0000_00FFh	8.5.9/253
303D_08BC	Memory Region Violation Status (RDC_MRVS11)	32	R/W	0000_0000h	8.5.10/254

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_08C0	Memory Region Start Address (RDC_MRSA12)	32	R/W	Undefined	8.5.7/250
303D_08C4	Memory Region End Address (RDC_MREA12)	32	R/W	Undefined	8.5.8/252
303D_08C8	Memory Region Control (RDC_MRC12)	32	R/W	0000_00FFh	8.5.9/253
303D_08CC	Memory Region Violation Status (RDC_MRVS12)	32	R/W	0000_0000h	8.5.10/254
303D_08D0	Memory Region Start Address (RDC_MRSA13)	32	R/W	Undefined	8.5.7/250
303D_08D4	Memory Region End Address (RDC_MREA13)	32	R/W	Undefined	8.5.8/252
303D_08D8	Memory Region Control (RDC_MRC13)	32	R/W	0000_00FFh	8.5.9/253
303D_08DC	Memory Region Violation Status (RDC_MRVS13)	32	R/W	0000_0000h	8.5.10/254
303D_08E0	Memory Region Start Address (RDC_MRSA14)	32	R/W	Undefined	8.5.7/250
303D_08E4	Memory Region End Address (RDC_MREA14)	32	R/W	Undefined	8.5.8/252
303D_08E8	Memory Region Control (RDC_MRC14)	32	R/W	0000_00FFh	8.5.9/253
303D_08EC	Memory Region Violation Status (RDC_MRVS14)	32	R/W	0000_0000h	8.5.10/254
303D_08F0	Memory Region Start Address (RDC_MRSA15)	32	R/W	Undefined	8.5.7/250
303D_08F4	Memory Region End Address (RDC_MREA15)	32	R/W	Undefined	8.5.8/252
303D_08F8	Memory Region Control (RDC_MRC15)	32	R/W	0000_00FFh	8.5.9/253
303D_08FC	Memory Region Violation Status (RDC_MRVS15)	32	R/W	0000_0000h	8.5.10/254
303D_0900	Memory Region Start Address (RDC_MRSA16)	32	R/W	Undefined	8.5.7/250
303D_0904	Memory Region End Address (RDC_MREA16)	32	R/W	Undefined	8.5.8/252
303D_0908	Memory Region Control (RDC_MRC16)	32	R/W	0000_00FFh	8.5.9/253
303D_090C	Memory Region Violation Status (RDC_MRVS16)	32	R/W	0000_0000h	8.5.10/254
303D_0910	Memory Region Start Address (RDC_MRSA17)	32	R/W	Undefined	8.5.7/250
303D_0914	Memory Region End Address (RDC_MREA17)	32	R/W	Undefined	8.5.8/252
303D_0918	Memory Region Control (RDC_MRC17)	32	R/W	0000_00FFh	8.5.9/253
303D_091C	Memory Region Violation Status (RDC_MRVS17)	32	R/W	0000_0000h	8.5.10/254
303D_0920	Memory Region Start Address (RDC_MRSA18)	32	R/W	Undefined	8.5.7/250
303D_0924	Memory Region End Address (RDC_MREA18)	32	R/W	Undefined	8.5.8/252
303D_0928	Memory Region Control (RDC_MRC18)	32	R/W	0000_00FFh	8.5.9/253
303D_092C	Memory Region Violation Status (RDC_MRVS18)	32	R/W	0000_0000h	8.5.10/254
303D_0930	Memory Region Start Address (RDC_MRSA19)	32	R/W	Undefined	8.5.7/250
303D_0934	Memory Region End Address (RDC_MREA19)	32	R/W	Undefined	8.5.8/252
303D_0938	Memory Region Control (RDC_MRC19)	32	R/W	0000_00FFh	8.5.9/253
303D_093C	Memory Region Violation Status (RDC_MRVS19)	32	R/W	0000_0000h	8.5.10/254
303D_0940	Memory Region Start Address (RDC_MRSA20)	32	R/W	Undefined	8.5.7/250
303D_0944	Memory Region End Address (RDC_MREA20)	32	R/W	Undefined	8.5.8/252
303D_0948	Memory Region Control (RDC_MRC20)	32	R/W	0000_00FFh	8.5.9/253
303D_094C	Memory Region Violation Status (RDC_MRVS20)	32	R/W	0000_0000h	8.5.10/254
303D_0950	Memory Region Start Address (RDC_MRSA21)	32	R/W	Undefined	8.5.7/250
303D_0954	Memory Region End Address (RDC_MREA21)	32	R/W	Undefined	8.5.8/252

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0958	Memory Region Control (RDC_MRC21)	32	R/W	0000_00FFh	8.5.9/253
303D_095C	Memory Region Violation Status (RDC_MRVS21)	32	R/W	0000_0000h	8.5.10/254
303D_0960	Memory Region Start Address (RDC_MRSA22)	32	R/W	Undefined	8.5.7/250
303D_0964	Memory Region End Address (RDC_MREA22)	32	R/W	Undefined	8.5.8/252
303D_0968	Memory Region Control (RDC_MRC22)	32	R/W	0000_00FFh	8.5.9/253
303D_096C	Memory Region Violation Status (RDC_MRVS22)	32	R/W	0000_0000h	8.5.10/254
303D_0970	Memory Region Start Address (RDC_MRSA23)	32	R/W	Undefined	8.5.7/250
303D_0974	Memory Region End Address (RDC_MREA23)	32	R/W	Undefined	8.5.8/252
303D_0978	Memory Region Control (RDC_MRC23)	32	R/W	0000_00FFh	8.5.9/253
303D_097C	Memory Region Violation Status (RDC_MRVS23)	32	R/W	0000_0000h	8.5.10/254
303D_0980	Memory Region Start Address (RDC_MRSA24)	32	R/W	Undefined	8.5.7/250
303D_0984	Memory Region End Address (RDC_MREA24)	32	R/W	Undefined	8.5.8/252
303D_0988	Memory Region Control (RDC_MRC24)	32	R/W	0000_00FFh	8.5.9/253
303D_098C	Memory Region Violation Status (RDC_MRVS24)	32	R/W	0000_0000h	8.5.10/254
303D_0990	Memory Region Start Address (RDC_MRSA25)	32	R/W	Undefined	8.5.7/250
303D_0994	Memory Region End Address (RDC_MREA25)	32	R/W	Undefined	8.5.8/252
303D_0998	Memory Region Control (RDC_MRC25)	32	R/W	0000_00FFh	8.5.9/253
303D_099C	Memory Region Violation Status (RDC_MRVS25)	32	R/W	0000_0000h	8.5.10/254
303D_09A0	Memory Region Start Address (RDC_MRSA26)	32	R/W	Undefined	8.5.7/250
303D_09A4	Memory Region End Address (RDC_MREA26)	32	R/W	Undefined	8.5.8/252
303D_09A8	Memory Region Control (RDC_MRC26)	32	R/W	0000_00FFh	8.5.9/253
303D_09AC	Memory Region Violation Status (RDC_MRVS26)	32	R/W	0000_0000h	8.5.10/254
303D_09B0	Memory Region Start Address (RDC_MRSA27)	32	R/W	Undefined	8.5.7/250
303D_09B4	Memory Region End Address (RDC_MREA27)	32	R/W	Undefined	8.5.8/252
303D_09B8	Memory Region Control (RDC_MRC27)	32	R/W	0000_00FFh	8.5.9/253
303D_09BC	Memory Region Violation Status (RDC_MRVS27)	32	R/W	0000_0000h	8.5.10/254
303D_09C0	Memory Region Start Address (RDC_MRSA28)	32	R/W	Undefined	8.5.7/250
303D_09C4	Memory Region End Address (RDC_MREA28)	32	R/W	Undefined	8.5.8/252
303D_09C8	Memory Region Control (RDC_MRC28)	32	R/W	0000_00FFh	8.5.9/253
303D_09CC	Memory Region Violation Status (RDC_MRVS28)	32	R/W	0000_0000h	8.5.10/254
303D_09D0	Memory Region Start Address (RDC_MRSA29)	32	R/W	Undefined	8.5.7/250
303D_09D4	Memory Region End Address (RDC_MREA29)	32	R/W	Undefined	8.5.8/252
303D_09D8	Memory Region Control (RDC_MRC29)	32	R/W	0000_00FFh	8.5.9/253
303D_09DC	Memory Region Violation Status (RDC_MRVS29)	32	R/W	0000_0000h	8.5.10/254
303D_09E0	Memory Region Start Address (RDC_MRSA30)	32	R/W	Undefined	8.5.7/250
303D_09E4	Memory Region End Address (RDC_MREA30)	32	R/W	Undefined	8.5.8/252
303D_09E8	Memory Region Control (RDC_MRC30)	32	R/W	0000_00FFh	8.5.9/253
303D_09EC	Memory Region Violation Status (RDC_MRVS30)	32	R/W	0000_0000h	8.5.10/254

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_09F0	Memory Region Start Address (RDC_MRSA31)	32	R/W	Undefined	8.5.7/250
303D_09F4	Memory Region End Address (RDC_MREA31)	32	R/W	Undefined	8.5.8/252
303D_09F8	Memory Region Control (RDC_MRC31)	32	R/W	0000_00FFh	8.5.9/253
303D_09FC	Memory Region Violation Status (RDC_MRVS31)	32	R/W	0000_0000h	8.5.10/254
303D_0A00	Memory Region Start Address (RDC_MRSA32)	32	R/W	Undefined	8.5.7/250
303D_0A04	Memory Region End Address (RDC_MREA32)	32	R/W	Undefined	8.5.8/252
303D_0A08	Memory Region Control (RDC_MRC32)	32	R/W	0000_00FFh	8.5.9/253
303D_0A0C	Memory Region Violation Status (RDC_MRVS32)	32	R/W	0000_0000h	8.5.10/254
303D_0A10	Memory Region Start Address (RDC_MRSA33)	32	R/W	Undefined	8.5.7/250
303D_0A14	Memory Region End Address (RDC_MREA33)	32	R/W	Undefined	8.5.8/252
303D_0A18	Memory Region Control (RDC_MRC33)	32	R/W	0000_00FFh	8.5.9/253
303D_0A1C	Memory Region Violation Status (RDC_MRVS33)	32	R/W	0000_0000h	8.5.10/254
303D_0A20	Memory Region Start Address (RDC_MRSA34)	32	R/W	Undefined	8.5.7/250
303D_0A24	Memory Region End Address (RDC_MREA34)	32	R/W	Undefined	8.5.8/252
303D_0A28	Memory Region Control (RDC_MRC34)	32	R/W	0000_00FFh	8.5.9/253
303D_0A2C	Memory Region Violation Status (RDC_MRVS34)	32	R/W	0000_0000h	8.5.10/254
303D_0A30	Memory Region Start Address (RDC_MRSA35)	32	R/W	Undefined	8.5.7/250
303D_0A34	Memory Region End Address (RDC_MREA35)	32	R/W	Undefined	8.5.8/252
303D_0A38	Memory Region Control (RDC_MRC35)	32	R/W	0000_00FFh	8.5.9/253
303D_0A3C	Memory Region Violation Status (RDC_MRVS35)	32	R/W	0000_0000h	8.5.10/254
303D_0A40	Memory Region Start Address (RDC_MRSA36)	32	R/W	Undefined	8.5.7/250
303D_0A44	Memory Region End Address (RDC_MREA36)	32	R/W	Undefined	8.5.8/252
303D_0A48	Memory Region Control (RDC_MRC36)	32	R/W	0000_00FFh	8.5.9/253
303D_0A4C	Memory Region Violation Status (RDC_MRVS36)	32	R/W	0000_0000h	8.5.10/254
303D_0A50	Memory Region Start Address (RDC_MRSA37)	32	R/W	Undefined	8.5.7/250
303D_0A54	Memory Region End Address (RDC_MREA37)	32	R/W	Undefined	8.5.8/252
303D_0A58	Memory Region Control (RDC_MRC37)	32	R/W	0000_00FFh	8.5.9/253
303D_0A5C	Memory Region Violation Status (RDC_MRVS37)	32	R/W	0000_0000h	8.5.10/254
303D_0A60	Memory Region Start Address (RDC_MRSA38)	32	R/W	Undefined	8.5.7/250
303D_0A64	Memory Region End Address (RDC_MREA38)	32	R/W	Undefined	8.5.8/252
303D_0A68	Memory Region Control (RDC_MRC38)	32	R/W	0000_00FFh	8.5.9/253
303D_0A6C	Memory Region Violation Status (RDC_MRVS38)	32	R/W	0000_0000h	8.5.10/254
303D_0A70	Memory Region Start Address (RDC_MRSA39)	32	R/W	Undefined	8.5.7/250
303D_0A74	Memory Region End Address (RDC_MREA39)	32	R/W	Undefined	8.5.8/252
303D_0A78	Memory Region Control (RDC_MRC39)	32	R/W	0000_00FFh	8.5.9/253
303D_0A7C	Memory Region Violation Status (RDC_MRVS39)	32	R/W	0000_0000h	8.5.10/254
303D_0A80	Memory Region Start Address (RDC_MRSA40)	32	R/W	Undefined	8.5.7/250
303D_0A84	Memory Region End Address (RDC_MREA40)	32	R/W	Undefined	8.5.8/252

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0A88	Memory Region Control (RDC_MRC40)	32	R/W	0000_00FFh	8.5.9/253
303D_0A8C	Memory Region Violation Status (RDC_MRVS40)	32	R/W	0000_0000h	8.5.10/254
303D_0A90	Memory Region Start Address (RDC_MRSA41)	32	R/W	Undefined	8.5.7/250
303D_0A94	Memory Region End Address (RDC_MREA41)	32	R/W	Undefined	8.5.8/252
303D_0A98	Memory Region Control (RDC_MRC41)	32	R/W	0000_00FFh	8.5.9/253
303D_0A9C	Memory Region Violation Status (RDC_MRVS41)	32	R/W	0000_0000h	8.5.10/254
303D_0AA0	Memory Region Start Address (RDC_MRSA42)	32	R/W	Undefined	8.5.7/250
303D_0AA4	Memory Region End Address (RDC_MREA42)	32	R/W	Undefined	8.5.8/252
303D_0AA8	Memory Region Control (RDC_MRC42)	32	R/W	0000_00FFh	8.5.9/253
303D_0AAC	Memory Region Violation Status (RDC_MRVS42)	32	R/W	0000_0000h	8.5.10/254
303D_0AB0	Memory Region Start Address (RDC_MRSA43)	32	R/W	Undefined	8.5.7/250
303D_0AB4	Memory Region End Address (RDC_MREA43)	32	R/W	Undefined	8.5.8/252
303D_0AB8	Memory Region Control (RDC_MRC43)	32	R/W	0000_00FFh	8.5.9/253
303D_0ABC	Memory Region Violation Status (RDC_MRVS43)	32	R/W	0000_0000h	8.5.10/254
303D_0AC0	Memory Region Start Address (RDC_MRSA44)	32	R/W	Undefined	8.5.7/250
303D_0AC4	Memory Region End Address (RDC_MREA44)	32	R/W	Undefined	8.5.8/252
303D_0AC8	Memory Region Control (RDC_MRC44)	32	R/W	0000_00FFh	8.5.9/253
303D_0ACC	Memory Region Violation Status (RDC_MRVS44)	32	R/W	0000_0000h	8.5.10/254
303D_0AD0	Memory Region Start Address (RDC_MRSA45)	32	R/W	Undefined	8.5.7/250
303D_0AD4	Memory Region End Address (RDC_MREA45)	32	R/W	Undefined	8.5.8/252
303D_0AD8	Memory Region Control (RDC_MRC45)	32	R/W	0000_00FFh	8.5.9/253
303D_0ADC	Memory Region Violation Status (RDC_MRVS45)	32	R/W	0000_0000h	8.5.10/254
303D_0AE0	Memory Region Start Address (RDC_MRSA46)	32	R/W	Undefined	8.5.7/250
303D_0AE4	Memory Region End Address (RDC_MREA46)	32	R/W	Undefined	8.5.8/252
303D_0AE8	Memory Region Control (RDC_MRC46)	32	R/W	0000_00FFh	8.5.9/253
303D_0AEC	Memory Region Violation Status (RDC_MRVS46)	32	R/W	0000_0000h	8.5.10/254
303D_0AF0	Memory Region Start Address (RDC_MRSA47)	32	R/W	Undefined	8.5.7/250
303D_0AF4	Memory Region End Address (RDC_MREA47)	32	R/W	Undefined	8.5.8/252
303D_0AF8	Memory Region Control (RDC_MRC47)	32	R/W	0000_00FFh	8.5.9/253
303D_0AFC	Memory Region Violation Status (RDC_MRVS47)	32	R/W	0000_0000h	8.5.10/254
303D_0B00	Memory Region Start Address (RDC_MRSA48)	32	R/W	Undefined	8.5.7/250
303D_0B04	Memory Region End Address (RDC_MREA48)	32	R/W	Undefined	8.5.8/252
303D_0B08	Memory Region Control (RDC_MRC48)	32	R/W	0000_00FFh	8.5.9/253
303D_0B0C	Memory Region Violation Status (RDC_MRVS48)	32	R/W	0000_0000h	8.5.10/254
303D_0B10	Memory Region Start Address (RDC_MRSA49)	32	R/W	Undefined	8.5.7/250
303D_0B14	Memory Region End Address (RDC_MREA49)	32	R/W	Undefined	8.5.8/252
303D_0B18	Memory Region Control (RDC_MRC49)	32	R/W	0000_00FFh	8.5.9/253
303D_0B1C	Memory Region Violation Status (RDC_MRVS49)	32	R/W	0000_0000h	8.5.10/254

Table continues on the next page...

RDC memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303D_0B20	Memory Region Start Address (RDC_MRSA50)	32	R/W	Undefined	8.5.7/250
303D_0B24	Memory Region End Address (RDC_MREA50)	32	R/W	Undefined	8.5.8/252
303D_0B28	Memory Region Control (RDC_MRC50)	32	R/W	0000_00FFh	8.5.9/253
303D_0B2C	Memory Region Violation Status (RDC_MRVS50)	32	R/W	0000_0000h	8.5.10/254
303D_0B30	Memory Region Start Address (RDC_MRSA51)	32	R/W	Undefined	8.5.7/250
303D_0B34	Memory Region End Address (RDC_MREA51)	32	R/W	Undefined	8.5.8/252
303D_0B38	Memory Region Control (RDC_MRC51)	32	R/W	0000_00FFh	8.5.9/253
303D_0B3C	Memory Region Violation Status (RDC_MRVS51)	32	R/W	0000_0000h	8.5.10/254

8.5.1 Version Information (RDC_VIR)

The VIR provides version information including the number of domains, number of master slots, number of peripheral slots, and number of memory regions.

Address: 303D_0000h base + 0h offset = 303D_0000h

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				NRGN								NPER								NMSTR								NDID			
W																																
Reset	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0

RDC_VIR field descriptions

Field	Description
31–28 Reserved	This field is reserved.
27–20 NRGN	Number of Memory Regions Indicates the number of memory regions in this instance of the RDC.
19–12 NPER	Number of Peripherals Indicates the number of peripherals that can be isolated or safe-shared
11–4 NMSTR	Number of Masters Indicates the number of masters supported by this instance of RDC.
NDID	Number of Domains Indicates the number of domain ids supported by this instance of the RDC. Add one to the register value to get the actual number of domains.

8.5.2 Status (RDC_STAT)

Address: 303D_0000h base + 24h offset = 303D_0024h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
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	Reserved							PDS	Reserved				DID			
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

RDC_STAT field descriptions

Field	Description
31–9 Reserved	This field is reserved.
8 PDS	Power Domain Status Indicates if the "Power Down" memory regions are powered and available. Power Down memory regions are only those memory regions susceptible to power outage for power savings are unavailable if this is zero. "Always-On" memory regions remain available. Always On memory regions are those regions that are not powered down unless the entire SoC is powered down. This signal remains low until all access controls have been restored to the domain. 0 Power Down Domain is OFF 1 Power Down Domain is ON
7–4 Reserved	This field is reserved.
DID	Domain ID The Domain ID of the core or bus master that is reading this. The value is different for requests from different domains.

8.5.3 Interrupt and Control (RDC_INTCTRL)

Address: 303D_0000h base + 28h offset = 303D_0028h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
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	Reserved															RCI_EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RDC_INTCTRL field descriptions

Field	Description
31–1 Reserved	This field is reserved.
0 RCI_EN	Restoration Complete Interrupt Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions. 0 Interrupt Disabled 1 Interrupt Enabled

8.5.4 Interrupt Status (RDC_INTSTAT)

Indication of Interrupt Pending for State Restoration

Address: 303D_0000h base + 2Ch offset = 303D_002Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
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															INT
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RDC_INTSTAT field descriptions

Field	Description
31–1 Reserved	This field is reserved.
0 INT	Interrupt Status Indicates state of interrupt signal for state restoration. This is that status of the interrupt enabled in RDC_INTCTRL. Write one to interrupt status to clear it. 0 No Interrupt Pending 1 Interrupt Pending

8.5.5 Master Domain Assignment (RDC_MDA*n*)

Address: 303D_0000h base + 200h offset + (4d × i), where i=0d to 26d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LCK	Reserved														
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	Reserved														DID	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RDC_MDA*n* field descriptions

Field	Description
31 LCK	0 Not Locked 1 Locked
30–2 Reserved	This field is reserved.
DID	Domain ID Indicates the domain to which the Master is assigned 00 Master assigned to Processing Domain 0 01 Master assigned to Processing Domain 1 10 Master assigned to Processing Domain 2 11 Master assigned to Processing Domain 3

8.5.6 Peripheral Domain Access Permissions (RDC_PDAP*n*)

Address: 303D_0000h base + 400h offset + (4d × i), where i=0d to 117d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LCK	SREQ	Reserved													
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	Reserved								D3R	D3W	D2R	D2W	D1R	D1W	D0R	D0W
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

RDC_PDAP*n* field descriptions

Field	Description
31 LCK	Peripheral Permissions Lock When set prevents further modification of the Peripheral Domain Access Permissions (sticky bit until reset) 0 Not Locked 1 Locked
30 SREQ	Semaphore Required When set the hardware semaphore state enforces the semaphore lock. If a domain has access permissions and a semaphore has locked a shared peripheral then only the domain holding the semaphore signal can access this peripheral. 0 Semaphores have no effect 1 Semaphores are enforced
29–8 Reserved	This field is reserved.
7 D3R	Domain 3 Read Access 0 No Read Access 1 Read Access Allowed
6 D3W	Domain 3 Write Access 0 No Write Access 1 Write Access Allowed
5 D2R	Domain 2 Read Access 0 No Read Access 1 Read Access Allowed
4 D2W	Domain 2 Write Access 0 No Write Access 1 Write Access Allowed
3 D1R	Domain 1 Read Access

Table continues on the next page...

RDC_PDAP*n* field descriptions (continued)

Field	Description
	0 No Read Access 1 Read Access Allowed
2 D1W	Domain 1 Write Access 0 No Write Access 1 Write Access Allowed
1 D0R	Domain 0 Read Access 0 No Read Access 1 Read Access Allowed
0 D0W	Domain 0 Write Access 0 No Write Access 1 Write Access Allowed

8.5.7 Memory Region Start Address (RDC_MRSA*n*)

NOTE

The DDR space is 33-bit width. The RDC memory region registers are 32-bit width. The RDC configuration is the most significant bits in the DDR address space (32:1). To set the start address for this configuration, the MRSA value should be shifted 1-bit and added to the DDR base address. The example below illustrates how to calculate the proper start and end address value. Please refer to the Memory Map for the actual DDR base address.

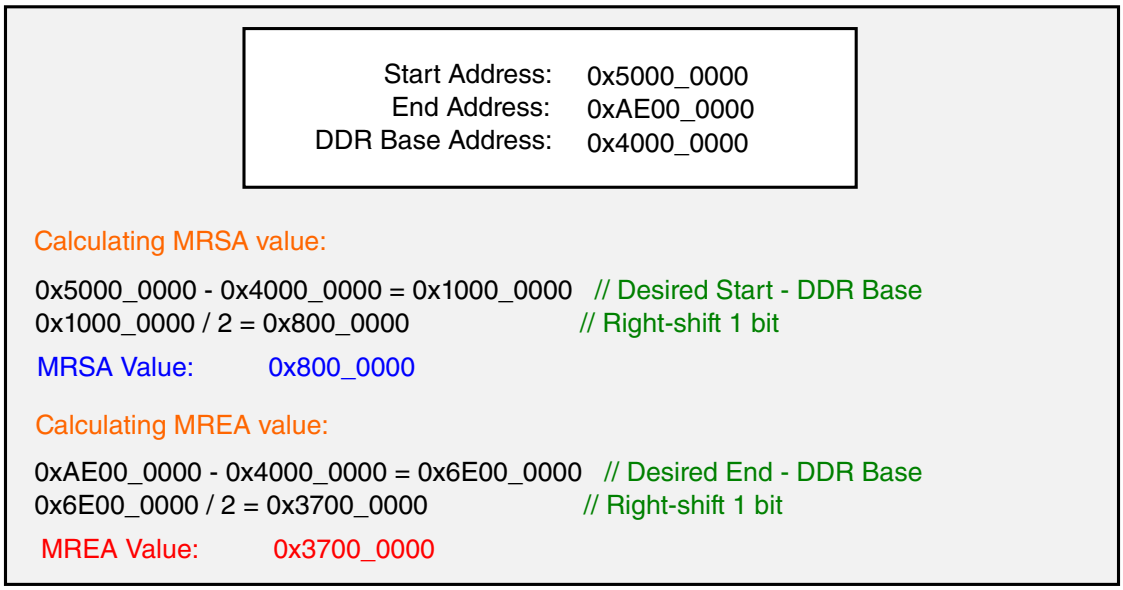


Figure 8-5. Calculating Address Value Examble

Address: 303D_0000h base + 800h offset + (16d × i), where i=0d to 51d

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	SADR																Reserved															
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

RDC_MRSA*n* field descriptions

Field	Description
31–7 SADR	Start address for memory region Lower bound (inclusive) modulo the defined granularity byte size of a region. The region size (granularity) is defined for each Memory/Port in the Memory Region Map section. Region boundaries are aligned to the minimum possible region size for the Memory/Port.
Reserved	This field is reserved.

8.5.8 Memory Region End Address (RDC_MREA*n*)

NOTE

The DDR space is 33-bit width. The RDC memory region registers are 32-bit width. The RDC configuration is the most significant bits in the DDR address space (32:1). To set the start address for this configuration, the MRSA value should be shifted 1-bit and added to the DDR base address. The example below illustrates how to calculate the proper start and end address value. Please refer to the Memory Map for the actual DDR base address.

```
Start Address:      0x5000_0000
End Address:        0xAE00_0000
DDR Base Address:   0x4000_0000

Calculating MRSA value:

0x5000_0000 - 0x4000_0000 = 0x1000_0000   // Desired Start - DDR Base
0x1000_0000 / 2 = 0x800_0000              // Right-shift 1 bit

MRSA Value:        0x800_0000

Calculating MREA value:

0xAE00_0000 - 0x4000_0000 = 0x6E00_0000   // Desired End - DDR Base
0x6E00_0000 / 2 = 0x3700_0000             // Right-shift 1 bit

MREA Value:        0x3700_0000
```

Figure 8-6. Calculating Address Value Examble

Address: 303D_0000h base + 804h offset + (16d × i), where i=0d to 51d

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	EADR																Reserved															
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

- x = Undefined at reset.

RDC_MREA*n* field descriptions

Field	Description
31–7 EADR	Upper bound for memory region

Table continues on the next page...

RDC_MREA*n* field descriptions (continued)

Field	Description
	Upper bound (exclusive) modulo the defined granularity byte size of a region. The region size (granularity) is defined for each Memory/Port in the Memory Region Map section. Region boundaries are aligned to the minimum possible region size for the Memory/Port.
Reserved	This field is reserved.

8.5.9 Memory Region Control (RDC_MRC*n*)

Address: 303D_0000h base + 808h offset + (16d × i), where i=0d to 51d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LCK	ENA	Reserved													
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	Reserved								D3R	D3W	D2R	D2W	D1R	D1W	D0R	D0W
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

RDC_MRC*n* field descriptions

Field	Description
31 LCK	Region Lock Locks all region fields from further modification except ENA, which can be set but not reset after LCK is set. LCK is a sticky bit. 0 No Lock. All fields in this register may be modified. 1 Locked. No fields in this register may be modified except ENA, which may be set but not cleared.
30 ENA	Region Enable Activates the memory region. If the region is not activated then the permissions and address boundaries have not affect and the region will be fully accessible. 0 Memory region is not defined or restricted. 1 Memory boundaries, domain permissions and controls are in effect.
29–8 Reserved	This field is reserved.
7 D3R	Domain 3 Read Access to Region 0 Processing Domain 3 does not have Read access to the memory region 1 Processing Domain 3 has Read access to the memory region
6 D3W	Domain 3 Write Access to Region 0 Processing Domain 3 does not have Write access to the memory region 1 Processing Domain 3 has Read access to the memory region
5 D2R	Domain 2 Read Access to Region

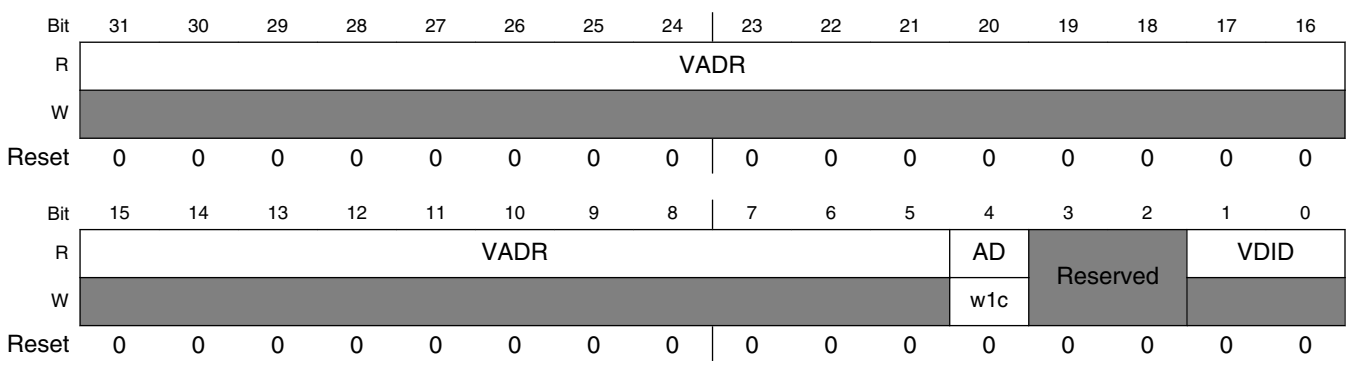
Table continues on the next page...

RDC_MRC*n* field descriptions (continued)

Field	Description
	0 Processing Domain 2 does not have Read access to the memory region 1 Processing Domain 2 has Read access to the memory region
4 D2W	Domain 2 Write Access to Region 0 Processing Domain 2 does not have Write access to the memory region 1 Processing Domain 2 has Write access to the memory region
3 D1R	Domain 1 Read Access to Region 0 Processing Domain 1 does not have Read access to the memory region 1 Processing Domain 1 has Read access to the memory region
2 D1W	Domain 1 Write Access to Region 0 Processing Domain 1 does not have Write access to the memory region 1 Processing Domain 1 has Write access to the memory region
1 D0R	Domain 0 Read Access to Region 0 Processing Domain 0 does not have Read access to the memory region 1 Processing Domain 0 has Read access to the memory region
0 D0W	Domain 0 Write Access to Region 0 Processing Domain 0 does not have Write access to the memory region 1 Processing Domain 0 has Write access to the memory region

8.5.10 Memory Region Violation Status (RDC_MRVS*n*)

Address: 303D_0000h base + 80Ch offset + (16d × i), where i=0d to 51d



RDC_MRVS*n* field descriptions

Field	Description
31–5 VADR	Violating Address The address of the denied access. The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID (s).

Table continues on the next page...

RDC_MRVS*n* field descriptions (continued)

Field	Description
4 AD	Access Denied Access to a memory region denied. This bit is cleared when this bit is written by one of the allowed domains.
3–2 Reserved	This field is reserved.
VDID	Violating Domain ID The domain ID of the denied access. The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. 00 Processing Domain 0 01 Processing Domain 1 10 Processing Domain 2 11 Processing Domain 3

8.6 RDC SEMA42 Memory Map/Register Definition

Only Supervisor Mode accesses are allowed on these registers. User accesses generate an error termination.

RDC_SEMAPHORE memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303B_0000	Gate Register (RDC_SEMAPHORE1_GATE0)	8	R/W	00h	8.6.1/259
303B_0001	Gate Register (RDC_SEMAPHORE1_GATE1)	8	R/W	00h	8.6.1/259
303B_0002	Gate Register (RDC_SEMAPHORE1_GATE2)	8	R/W	00h	8.6.1/259
303B_0003	Gate Register (RDC_SEMAPHORE1_GATE3)	8	R/W	00h	8.6.1/259
303B_0004	Gate Register (RDC_SEMAPHORE1_GATE4)	8	R/W	00h	8.6.1/259
303B_0005	Gate Register (RDC_SEMAPHORE1_GATE5)	8	R/W	00h	8.6.1/259
303B_0006	Gate Register (RDC_SEMAPHORE1_GATE6)	8	R/W	00h	8.6.1/259
303B_0007	Gate Register (RDC_SEMAPHORE1_GATE7)	8	R/W	00h	8.6.1/259
303B_0008	Gate Register (RDC_SEMAPHORE1_GATE8)	8	R/W	00h	8.6.1/259
303B_0009	Gate Register (RDC_SEMAPHORE1_GATE9)	8	R/W	00h	8.6.1/259
303B_000A	Gate Register (RDC_SEMAPHORE1_GATE10)	8	R/W	00h	8.6.1/259
303B_000B	Gate Register (RDC_SEMAPHORE1_GATE11)	8	R/W	00h	8.6.1/259
303B_000C	Gate Register (RDC_SEMAPHORE1_GATE12)	8	R/W	00h	8.6.1/259
303B_000D	Gate Register (RDC_SEMAPHORE1_GATE13)	8	R/W	00h	8.6.1/259
303B_000E	Gate Register (RDC_SEMAPHORE1_GATE14)	8	R/W	00h	8.6.1/259

Table continues on the next page...

RDC_SEMAPHORE memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303B_000F	Gate Register (RDC_SEMAPHORE1_GATE15)	8	R/W	00h	8.6.1/259
303B_0010	Gate Register (RDC_SEMAPHORE1_GATE16)	8	R/W	00h	8.6.1/259
303B_0011	Gate Register (RDC_SEMAPHORE1_GATE17)	8	R/W	00h	8.6.1/259
303B_0012	Gate Register (RDC_SEMAPHORE1_GATE18)	8	R/W	00h	8.6.1/259
303B_0013	Gate Register (RDC_SEMAPHORE1_GATE19)	8	R/W	00h	8.6.1/259
303B_0014	Gate Register (RDC_SEMAPHORE1_GATE20)	8	R/W	00h	8.6.1/259
303B_0015	Gate Register (RDC_SEMAPHORE1_GATE21)	8	R/W	00h	8.6.1/259
303B_0016	Gate Register (RDC_SEMAPHORE1_GATE22)	8	R/W	00h	8.6.1/259
303B_0017	Gate Register (RDC_SEMAPHORE1_GATE23)	8	R/W	00h	8.6.1/259
303B_0018	Gate Register (RDC_SEMAPHORE1_GATE24)	8	R/W	00h	8.6.1/259
303B_0019	Gate Register (RDC_SEMAPHORE1_GATE25)	8	R/W	00h	8.6.1/259
303B_001A	Gate Register (RDC_SEMAPHORE1_GATE26)	8	R/W	00h	8.6.1/259
303B_001B	Gate Register (RDC_SEMAPHORE1_GATE27)	8	R/W	00h	8.6.1/259
303B_001C	Gate Register (RDC_SEMAPHORE1_GATE28)	8	R/W	00h	8.6.1/259
303B_001D	Gate Register (RDC_SEMAPHORE1_GATE29)	8	R/W	00h	8.6.1/259
303B_001E	Gate Register (RDC_SEMAPHORE1_GATE30)	8	R/W	00h	8.6.1/259
303B_001F	Gate Register (RDC_SEMAPHORE1_GATE31)	8	R/W	00h	8.6.1/259
303B_0020	Gate Register (RDC_SEMAPHORE1_GATE32)	8	R/W	00h	8.6.1/259
303B_0021	Gate Register (RDC_SEMAPHORE1_GATE33)	8	R/W	00h	8.6.1/259
303B_0022	Gate Register (RDC_SEMAPHORE1_GATE34)	8	R/W	00h	8.6.1/259
303B_0023	Gate Register (RDC_SEMAPHORE1_GATE35)	8	R/W	00h	8.6.1/259
303B_0024	Gate Register (RDC_SEMAPHORE1_GATE36)	8	R/W	00h	8.6.1/259
303B_0025	Gate Register (RDC_SEMAPHORE1_GATE37)	8	R/W	00h	8.6.1/259
303B_0026	Gate Register (RDC_SEMAPHORE1_GATE38)	8	R/W	00h	8.6.1/259
303B_0027	Gate Register (RDC_SEMAPHORE1_GATE39)	8	R/W	00h	8.6.1/259
303B_0028	Gate Register (RDC_SEMAPHORE1_GATE40)	8	R/W	00h	8.6.1/259
303B_0029	Gate Register (RDC_SEMAPHORE1_GATE41)	8	R/W	00h	8.6.1/259
303B_002A	Gate Register (RDC_SEMAPHORE1_GATE42)	8	R/W	00h	8.6.1/259
303B_002B	Gate Register (RDC_SEMAPHORE1_GATE43)	8	R/W	00h	8.6.1/259
303B_002C	Gate Register (RDC_SEMAPHORE1_GATE44)	8	R/W	00h	8.6.1/259
303B_002D	Gate Register (RDC_SEMAPHORE1_GATE45)	8	R/W	00h	8.6.1/259
303B_002E	Gate Register (RDC_SEMAPHORE1_GATE46)	8	R/W	00h	8.6.1/259
303B_002F	Gate Register (RDC_SEMAPHORE1_GATE47)	8	R/W	00h	8.6.1/259
303B_0030	Gate Register (RDC_SEMAPHORE1_GATE48)	8	R/W	00h	8.6.1/259
303B_0031	Gate Register (RDC_SEMAPHORE1_GATE49)	8	R/W	00h	8.6.1/259
303B_0032	Gate Register (RDC_SEMAPHORE1_GATE50)	8	R/W	00h	8.6.1/259
303B_0033	Gate Register (RDC_SEMAPHORE1_GATE51)	8	R/W	00h	8.6.1/259
303B_0034	Gate Register (RDC_SEMAPHORE1_GATE52)	8	R/W	00h	8.6.1/259

Table continues on the next page...

RDC_SEMAPHORE memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303B_0035	Gate Register (RDC_SEMAPHORE1_GATE53)	8	R/W	00h	8.6.1/259
303B_0036	Gate Register (RDC_SEMAPHORE1_GATE54)	8	R/W	00h	8.6.1/259
303B_0037	Gate Register (RDC_SEMAPHORE1_GATE55)	8	R/W	00h	8.6.1/259
303B_0038	Gate Register (RDC_SEMAPHORE1_GATE56)	8	R/W	00h	8.6.1/259
303B_0039	Gate Register (RDC_SEMAPHORE1_GATE57)	8	R/W	00h	8.6.1/259
303B_003A	Gate Register (RDC_SEMAPHORE1_GATE58)	8	R/W	00h	8.6.1/259
303B_003B	Gate Register (RDC_SEMAPHORE1_GATE59)	8	R/W	00h	8.6.1/259
303B_003C	Gate Register (RDC_SEMAPHORE1_GATE60)	8	R/W	00h	8.6.1/259
303B_003D	Gate Register (RDC_SEMAPHORE1_GATE61)	8	R/W	00h	8.6.1/259
303B_003E	Gate Register (RDC_SEMAPHORE1_GATE62)	8	R/W	00h	8.6.1/259
303B_003F	Gate Register (RDC_SEMAPHORE1_GATE63)	8	R/W	00h	8.6.1/259
303B_0040	Reset Gate Write (RDC_SEMAPHORE1_RSTGT_W)	16	R/W	0000h	8.6.2/260
303B_0040	Reset Gate Read (RDC_SEMAPHORE1_RSTGT_R)	16	R/W	0000h	8.6.3/261
303C_0000	Gate Register (RDC_SEMAPHORE2_GATE0)	8	R/W	00h	8.6.1/259
303C_0001	Gate Register (RDC_SEMAPHORE2_GATE1)	8	R/W	00h	8.6.1/259
303C_0002	Gate Register (RDC_SEMAPHORE2_GATE2)	8	R/W	00h	8.6.1/259
303C_0003	Gate Register (RDC_SEMAPHORE2_GATE3)	8	R/W	00h	8.6.1/259
303C_0004	Gate Register (RDC_SEMAPHORE2_GATE4)	8	R/W	00h	8.6.1/259
303C_0005	Gate Register (RDC_SEMAPHORE2_GATE5)	8	R/W	00h	8.6.1/259
303C_0006	Gate Register (RDC_SEMAPHORE2_GATE6)	8	R/W	00h	8.6.1/259
303C_0007	Gate Register (RDC_SEMAPHORE2_GATE7)	8	R/W	00h	8.6.1/259
303C_0008	Gate Register (RDC_SEMAPHORE2_GATE8)	8	R/W	00h	8.6.1/259
303C_0009	Gate Register (RDC_SEMAPHORE2_GATE9)	8	R/W	00h	8.6.1/259
303C_000A	Gate Register (RDC_SEMAPHORE2_GATE10)	8	R/W	00h	8.6.1/259
303C_000B	Gate Register (RDC_SEMAPHORE2_GATE11)	8	R/W	00h	8.6.1/259
303C_000C	Gate Register (RDC_SEMAPHORE2_GATE12)	8	R/W	00h	8.6.1/259
303C_000D	Gate Register (RDC_SEMAPHORE2_GATE13)	8	R/W	00h	8.6.1/259
303C_000E	Gate Register (RDC_SEMAPHORE2_GATE14)	8	R/W	00h	8.6.1/259
303C_000F	Gate Register (RDC_SEMAPHORE2_GATE15)	8	R/W	00h	8.6.1/259
303C_0010	Gate Register (RDC_SEMAPHORE2_GATE16)	8	R/W	00h	8.6.1/259
303C_0011	Gate Register (RDC_SEMAPHORE2_GATE17)	8	R/W	00h	8.6.1/259
303C_0012	Gate Register (RDC_SEMAPHORE2_GATE18)	8	R/W	00h	8.6.1/259
303C_0013	Gate Register (RDC_SEMAPHORE2_GATE19)	8	R/W	00h	8.6.1/259
303C_0014	Gate Register (RDC_SEMAPHORE2_GATE20)	8	R/W	00h	8.6.1/259
303C_0015	Gate Register (RDC_SEMAPHORE2_GATE21)	8	R/W	00h	8.6.1/259
303C_0016	Gate Register (RDC_SEMAPHORE2_GATE22)	8	R/W	00h	8.6.1/259
303C_0017	Gate Register (RDC_SEMAPHORE2_GATE23)	8	R/W	00h	8.6.1/259
303C_0018	Gate Register (RDC_SEMAPHORE2_GATE24)	8	R/W	00h	8.6.1/259

Table continues on the next page...

RDC_SEMAPHORE memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303C_0019	Gate Register (RDC_SEMAPHORE2_GATE25)	8	R/W	00h	8.6.1/259
303C_001A	Gate Register (RDC_SEMAPHORE2_GATE26)	8	R/W	00h	8.6.1/259
303C_001B	Gate Register (RDC_SEMAPHORE2_GATE27)	8	R/W	00h	8.6.1/259
303C_001C	Gate Register (RDC_SEMAPHORE2_GATE28)	8	R/W	00h	8.6.1/259
303C_001D	Gate Register (RDC_SEMAPHORE2_GATE29)	8	R/W	00h	8.6.1/259
303C_001E	Gate Register (RDC_SEMAPHORE2_GATE30)	8	R/W	00h	8.6.1/259
303C_001F	Gate Register (RDC_SEMAPHORE2_GATE31)	8	R/W	00h	8.6.1/259
303C_0020	Gate Register (RDC_SEMAPHORE2_GATE32)	8	R/W	00h	8.6.1/259
303C_0021	Gate Register (RDC_SEMAPHORE2_GATE33)	8	R/W	00h	8.6.1/259
303C_0022	Gate Register (RDC_SEMAPHORE2_GATE34)	8	R/W	00h	8.6.1/259
303C_0023	Gate Register (RDC_SEMAPHORE2_GATE35)	8	R/W	00h	8.6.1/259
303C_0024	Gate Register (RDC_SEMAPHORE2_GATE36)	8	R/W	00h	8.6.1/259
303C_0025	Gate Register (RDC_SEMAPHORE2_GATE37)	8	R/W	00h	8.6.1/259
303C_0026	Gate Register (RDC_SEMAPHORE2_GATE38)	8	R/W	00h	8.6.1/259
303C_0027	Gate Register (RDC_SEMAPHORE2_GATE39)	8	R/W	00h	8.6.1/259
303C_0028	Gate Register (RDC_SEMAPHORE2_GATE40)	8	R/W	00h	8.6.1/259
303C_0029	Gate Register (RDC_SEMAPHORE2_GATE41)	8	R/W	00h	8.6.1/259
303C_002A	Gate Register (RDC_SEMAPHORE2_GATE42)	8	R/W	00h	8.6.1/259
303C_002B	Gate Register (RDC_SEMAPHORE2_GATE43)	8	R/W	00h	8.6.1/259
303C_002C	Gate Register (RDC_SEMAPHORE2_GATE44)	8	R/W	00h	8.6.1/259
303C_002D	Gate Register (RDC_SEMAPHORE2_GATE45)	8	R/W	00h	8.6.1/259
303C_002E	Gate Register (RDC_SEMAPHORE2_GATE46)	8	R/W	00h	8.6.1/259
303C_002F	Gate Register (RDC_SEMAPHORE2_GATE47)	8	R/W	00h	8.6.1/259
303C_0030	Gate Register (RDC_SEMAPHORE2_GATE48)	8	R/W	00h	8.6.1/259
303C_0031	Gate Register (RDC_SEMAPHORE2_GATE49)	8	R/W	00h	8.6.1/259
303C_0032	Gate Register (RDC_SEMAPHORE2_GATE50)	8	R/W	00h	8.6.1/259
303C_0033	Gate Register (RDC_SEMAPHORE2_GATE51)	8	R/W	00h	8.6.1/259
303C_0034	Gate Register (RDC_SEMAPHORE2_GATE52)	8	R/W	00h	8.6.1/259
303C_0035	Gate Register (RDC_SEMAPHORE2_GATE53)	8	R/W	00h	8.6.1/259
303C_0036	Gate Register (RDC_SEMAPHORE2_GATE54)	8	R/W	00h	8.6.1/259
303C_0037	Gate Register (RDC_SEMAPHORE2_GATE55)	8	R/W	00h	8.6.1/259
303C_0038	Gate Register (RDC_SEMAPHORE2_GATE56)	8	R/W	00h	8.6.1/259
303C_0039	Gate Register (RDC_SEMAPHORE2_GATE57)	8	R/W	00h	8.6.1/259
303C_003A	Gate Register (RDC_SEMAPHORE2_GATE58)	8	R/W	00h	8.6.1/259
303C_003B	Gate Register (RDC_SEMAPHORE2_GATE59)	8	R/W	00h	8.6.1/259
303C_003C	Gate Register (RDC_SEMAPHORE2_GATE60)	8	R/W	00h	8.6.1/259
303C_003D	Gate Register (RDC_SEMAPHORE2_GATE61)	8	R/W	00h	8.6.1/259
303C_003E	Gate Register (RDC_SEMAPHORE2_GATE62)	8	R/W	00h	8.6.1/259

Table continues on the next page...

RDC_SEMAPHORE memory map (continued)

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
303C_003F	Gate Register (RDC_SEMAPHORE2_GATE63)	8	R/W	00h	8.6.1/259
303C_0040	Reset Gate Write (RDC_SEMAPHORE2_RSTGT_W)	16	R/W	0000h	8.6.2/260
303C_0040	Reset Gate Read (RDC_SEMAPHORE2_RSTGT_R)	16	R/W	0000h	8.6.3/261

8.6.1 Gate Register (RDC_SEMAPHORE*x*_GATE*n*)

Each semaphore gate is implemented in a 4-bit finite state machine, right-justified in a byte data structure. The hardware uses the logical bus master number (master_index) in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. Once locked, a gate can (and must) be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate can be updated via a write operation at a time. Attempted writes with a data value that is neither the unlock value nor the appropriate lock value (master_index + 1) are simply treated as "no operation" and do not affect any gate state. Attempts to write multiple gates in a single aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Address: Base address + 0h offset + (1d × i), where i=0d to 63d

Bit	7	6	5	4	3	2	1	0
Read	0		LDOM		GTFSM			
Write								
Reset	0	0	0	0	0	0	0	0

RDC_SEMAPHORE*x*_GATE*n* field descriptions

Field	Description
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5–4 LDOM	Read-only bits. They indicate which domain had currently locked the gate. 00 The gate is locked by domain 0. (True if bits [3:0] do not equal 0000.) 01 The gate has been locked by domain 1. 10 The gate has been locked by domain 2. 11 The gate has been locked by domain 3.
GTFSM	Gate Finite State Machine. The state of the gate reflects the last processor that locked it, which can be useful during system debug. The hardware gate is maintained in a 16-state implementation, defined as:

Table continues on the next page...

RDC_SEMAPHORE*x*_GATE*n* field descriptions (continued)

Field	Description
	0000 The gate is unlocked (free). 0001 The gate has been locked by processor with master_index = 0. 0010 The gate has been locked by processor with master_index = 1. 0011 The gate has been locked by processor with master_index = 2. 0100 The gate has been locked by processor with master_index = 3. 0101 The gate has been locked by processor with master_index = 4. 0110 The gate has been locked by processor with master_index = 5. 0111 The gate has been locked by processor with master_index = 6. 1000 The gate has been locked by processor with master_index = 7. 1001 The gate has been locked by processor with master_index = 8. 1010 The gate has been locked by processor with master_index = 9. 1011 The gate has been locked by processor with master_index = 10. 1100 The gate has been locked by processor with master_index = 11. 1101 The gate has been locked by processor with master_index = 12. 1110 The gate has been locked by processor with master_index = 13. 1111 The gate has been locked by processor with master_index = 14.

8.6.2 Reset Gate Write (RDC_SEMAPHORE*x*_RSTGT_W)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the RDC Semaphores module implements a "secure" reset mechanism that allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the RDC_SEMA42RSTGT memory location. The least significant byte (RDC_SEMA42RSTGT[RSTGDP]) must be 0xE2; the most significant byte is a "don't_care" for this reference.
2. The same processor then performs a second 16-bit write to the RDC_SEMA42RSTGT location. For this write, the lower byte (RDC_SEMA42RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the upper byte (RDC_SEMA42RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared, or else that all gates are to be cleared. If the same processor writes incorrect data on the second access or another processor performs the second write access, the special gate reset sequence is aborted and no error signal will be asserted.

3. Reads of the RDC_SEMA42RSTGT location return information on the 2-bit state machine (RDC_SEMA42RSTGT[RSTGSM]) that implements this function, the bus master performing the reset (RDC_SEMA42RSTGT[RSTGMS]), and the gate number(s) last cleared (RDC_SEMA42RSTGT[RSTGTN]). Reads of the RDC_SEMA42RSTGT register do not affect the secure reset finite state machine in any manner.

Address: Base address + 40h offset

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	RSTGTN								0							
Write									RSTGDP							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RDC_SEMAPHOREx_RSTGT_W field descriptions

Field	Description
15–8 RSTGTN	Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write. If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates.
RSTGDP	Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xE2 while the second write requires RSTGDP = 0x1D.

8.6.3 Reset Gate Read (RDC_SEMAPHOREx_RSTGT_R)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the RDC Semaphores module implements a "secure" reset mechanism that allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the RDC_SEMA42RSTGT memory location. The least significant byte (RDC_SEMA42RSTGT[RSTGDP]) must be 0xE2; the most significant byte is a "don't_care" for this reference.
2. The same processor then performs a second 16-bit write to the RDC_SEMA42RSTGT location. For this write, the lower byte (RDC_SEMA42RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the upper byte (RDC_SEMA42RSTGT[RSTGTN]) specifies the

gate(s) to be reset. This gate field can specify a single gate be cleared, or else that all gates are to be cleared. If the same processor writes incorrect data on the second access or another processor performs the second write access, the special gate reset sequence is aborted and no error signal will be asserted.

3. Reads of the RDC_SEMA42RSTGT location return information on the 2-bit state machine (RDC_SEMA42RSTGT[RSTGSM]) that implements this function, the bus master performing the reset (RDC_SEMA42RSTGT[RSTGMS]), and the gate number(s) last cleared (RDC_SEMA42RSTGT[RSTGTN]). Reads of the RDC_SEMA42RSTGT register do not affect the secure reset finite state machine in any manner.

Address: Base address + 40h offset

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	RSTGTN								0		RSTGSM		RSTGMS			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RDC_SEMAPHOREx_RSTGT_R field descriptions

Field	Description
15–8 RSTGTN	Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write. If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates.
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5–4 RSTGSM	Reset Gate Finite State Machine. Reads of the RDC_SEMA42RSTGT register return the encoded state machine value. Note the RSTGSM = 10 state is valid for only a single machine cycle, so it is impossible for a read to return this value. The reset state machine is maintained in a 2-bit, 3-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. The "01" state persists for only one clock cycle. Software will never be able to observe this state. 11 This state encoding is never used and therefore reserved.
RSTGMS	Reset Gate Bus Master. This 4-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register must be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. The association between system bus master port numbers, the associated bus master device, and the logical processor number is SoC-specific. Consult the device reference manual for this information.

Chapter 9
TrustZone Address Space Controller (TZASC)

9.1 Overview

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

The TZASC block diagram is shown in figure below.

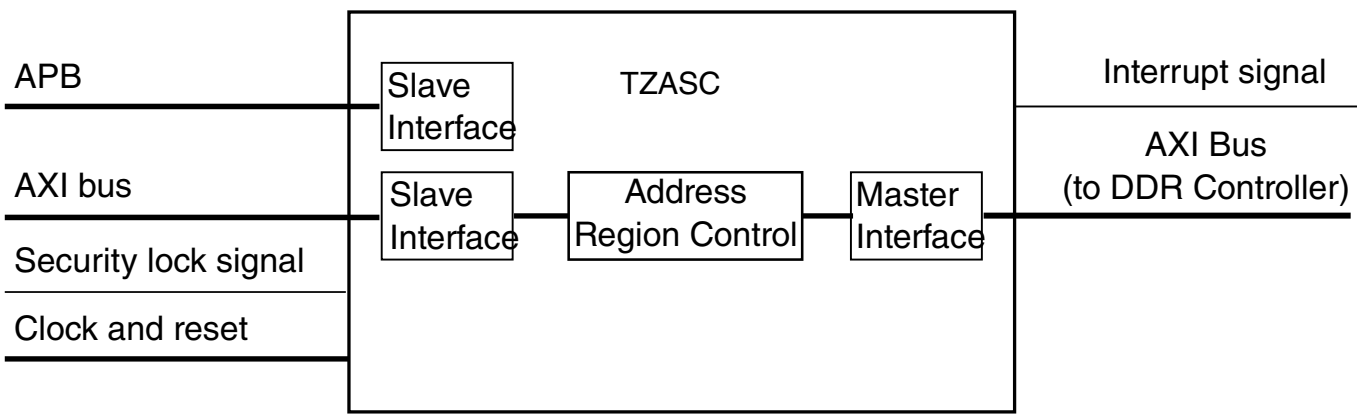


Figure 9-1. TZASC Block Diagram

The TZASC is an IP by Arm ("CoreLink™ TrustZone Address Space Controller TZC-380"), designed to provide configurable protection over program (SW) memory space.

The main features of TZASC are:

- Supports 16 independent address regions
- Access controls are independently programmable for each address region
- Sensitive registers may be locked
- Host interrupt may be programmed to signal attempted access control violations

- AXI master/slave interfaces for transactions
- APB slave interface for configuration and status reporting

NOTE

In this device it is necessary to set TZASC_ID_SWAP_BYPASS in IOMUXC_GPR10[1] to avoid an AXI bus error when using GPU. Further details about TZASC_ID_SWAP_BYPASS can be found in the device Reference Manual.

9.2 Clocks

The table found here describes the clock sources for TZASC.

Table 9-1. TZASC Clocks

Clock name	Clock Root	Description
aclk	ccm_clk_root	Module clock

9.3 Address Mapping in various memory mapping modes

The TZASC region base address starts at the beginning of DDR memory space (0x40000000) instead of the beginning of memory map (0x00000000). In this case the addresses configured in TZASC controller will be 1GB (0x40000000) offset and does not match the local addresses.

For example, setting region_setup_low_x=0xBE000000 maps DDR_ADDR=0xFE000000, the same behavior is observed with fail_address_x registers.

Memory "aliasing" implications on TZASC settings - in systems which does not utilize the maximal supported DDR space the controller is designed for, the whole DDR memory map becomes "aliased" (replicated) by the size of the physical memory used. In such cases, the TZASC must be configured to protect all aliased regions as well (i.e. effectively reducing the number of available TZASC regions, since all aliased regions must be handled, for each "real" space needing protection).

For complete details on TZASC functionality and the programming model, see the Arm document, “CoreLink™ TrustZone Address Space Controller TZC-380 Technical Reference Manual, (Rev r0p1 or newer)”, available at **http://infocenter.arm.com**.

Chapter 10
Cryptographic Acceleration and Assurance Module (CAAM)

10.1 Overview of CAAM (cryptographic acceleration and assurance module) functionality

CAAM is the chip's cryptographic acceleration and offloading hardware and combines cryptographic and other mathematical functions to create a modular and scalable hardware acceleration and assurance engine. CAAM implements the following functions:

- Block encryption algorithms
- Stream cipher algorithms
- Hashing algorithms
- Public key algorithms
- Run-time integrity checking
- Secure Memory controller
- Hardware random number generator

This version of CAAM also enables significant system-level performance improvements by providing higher-level cryptographic protocol operations.

CAAM includes the following interfaces:

- A Register interface for the processor to write configuration and command information, and to read status information
- A memory slave interface that gives access to the Secure Memory
- A DMA interface that allows CAAM to read/write data from external memory

- Job Queue Controller with 3 Job Rings
- 1 Descriptor Controller (DECO) :
  - Responsible for managing the sequencing, context, and execution of descriptors
  - Responsible for initating data transfers via the DMA interface
  - Responsible for managing keys and directing data to and from CHA(s)
  - Responsible for performing packet header and trailer processing as defined by the descriptor

- Run-Time Integrity Checker (RTIC)
- Crypto Hardware Accelerators (CHAs)
  - A Public Key Hardware Accelerator (PKHA)
  - A Random Number Generator (RNG)
  - An Advanced Encryption Standard Hardware Accelerator (AESA)
  - A Message Digest Hardware Accelerator (MDHA)
  - A Data Encryption Standard Hardware Accelerator (DESA)
  - An Alleged RC-4 Hardware Accelerator (AFHA)

This figure shows the block diagram for CAAM.

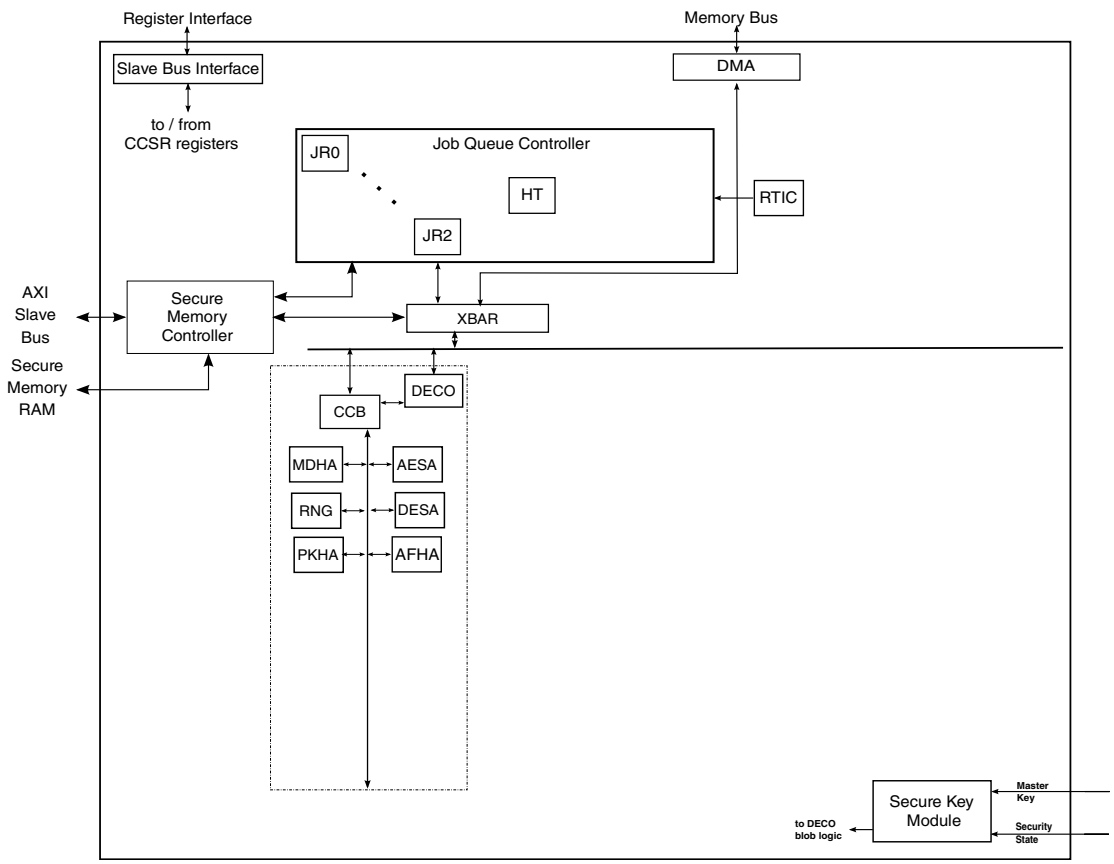


Figure 10-1. CAAM block diagram

10.2 Feature summary

CAAM includes the following features:

- SoC HW interfaces
  - A 32-bit slave bus register interface
  - A 32-bit data / 32-bit address DMA master bus interface

        - Automatic byte, half-word ordering of data read/written
        - Scatter/gather support for data
- Offloading of cryptographic functions via a programmable job descriptor language
    - Job Descriptors can contain multiple function commands.
    - Job Descriptors can be chained to additional Job Descriptors.
    - Job Descriptors can be submitted via 3 separate hardware-implemented Job Rings.
    - Each Job Ring implements access controls.
- Secure Memory
    - 32-bit AXI Slave Bus Interface
    - One default partition, plus 7 optional partitions
    - Each partition can be owned by any Job Ring owner
    - 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 DID value
        - 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
- Special-purpose cryptographic keys
    - 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 non-volatile 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
- Public key cryptography
  - Modular Arithmetic
    - Addition, subtraction, multiplication, exponentiation, reduction, inversion, greatest common denominator
    - Both integer and binary polynomial functions
    - Modulus size up to 4096 bits
    - Arithmetic operations performed with 32-bit-digit arithmetic unit
    - Timing-equalized and normal versions of modular exponentiation
    - Primality testing up to 4096 bits
  - DSA
    - DSA sign and verify
    - Verify with private key
    - DSA key generation
    - Non-timing-equalized versions of private-key operations
    - Timing-equalized versions of sign and key generation
    - Non-timing-equalized versions of sign and key generation
  - Diffie-Hellman
    - Diffie-Hellman (DH) key agreement
    - Key generation
    - Timing-equalized versions of key agreement and key generation
    - Non-timing-equalized versions of key agreement and key generation
  - RSA
    - Modulus size up to 4096 bits
    - Public and Private Key operations
    - Private keys in (n,d), (p,q,d), or 5-part (p,q,dp,dq,c) forms
    - Private Key operations (decrypt, sign) timing equalized to thwart side channel attack
    - Non-timing-equalized versions of private-key operations
  - Elliptic curve cryptography
    - Point add, point double, point multiply on both prime field and binary polynomial field curves
    - Point validation (is point on curve) both prime field and binary polynomial field curves
    - Timing-equalized and normal versions of point multiplication
    - Public Key validation
    - Elliptic curve digital signature algorithm (ECDSA) sign and verify
    - ECDSA verify with private key

        - Elliptic curve Diffie-Hellman key agreement
        - ECDSA and ECDH key generation
        - Modulus size up to 1024 bits
        - Timing-equalized versions of ECDSA sign and key generation
        - Non-timing-equalized versions of ECDSA sign and key generation
- Authentication
    - Hashing algorithms
        - MD5
        - SHA-1
        - SHA-224
        - SHA-256
    - Message authentication codes (MAC)
        - HMAC-all hashing algorithms
        - AES-CMAC
        - AES-XCBC-MAC
    - ICV checking
- Authenticated encryption algorithms (also known as AEAD algorithms)
    - AES-CCM (Counter with CBC-MAC)
    - AES-GCM (Galois counter mode)
- 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, OFB for both AES and DES block ciphers
        - CBC-CS2, CFB128 and CTR for AES
        - CFB8 for DES
- 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
    - For lower-power consumption, oscillator is off when not generating entropy
    - Designed to be NIST-compliant, pseudo random-number generator seeded using hardware-generated entropy
- Run-time integrity checking
    - SHA-256 message authentication
    - Segmented data-gathering to support non-contiguous data blocks in memory
    - Support for up to four independent memory blocks
- Advanced protocol support
    - Support for protocol-specific padding
- Virtualization features

- Black keys are cryptographically separated per security domain
- Blobs are cryptographically separated per security domain
- Trusted descriptors are cryptographically separated per security domain
- Secure Memory partitions are separated per security domain

10.3 CAAM implementation

CAAM is programmed using CAAM Job Descriptors that indicate the operations to be performed and that 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. The default configuration for this version of CAAM is little-endian mode.

10.3.1 CAAM submodules

The CAAM core contains the following submodules:

- Master bus interface
- Slave bus interface
- Register bus interface
- Job Queue Controller (JQC)
- Run-Time Integrity Checker (RTIC)
- Descriptor Controller (DECO)
- CHA control block (CCB)
- Cryptographic hardware accelerators (CHAs)
  - Public key hardware accelerator (PKHA)
  - Data encryption standard (DES) accelerator (DESA)
  - Advanced encryption standard (AES) accelerator (AESA)
  - Message digest (hashing) hardware accelerator (MDHA)
  - Random-number generator (RNG)
  - ARC four (alleged RC4) hardware accelerator (AFHA)
- 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 one or more CHAs, DECO communicates with the CHAs by means of the CCB.

10.3.2 CAAM Versions with Encryption Disabled

In some situations it may be desirable to disable the encryption algorithms implemented in CAAM. When the CAAM clock disable fuse is blown, the clock to CAAM is permanently disabled. In this case no CAAM operations are possible.When the CAAM encryption disable fuse is blown, all the CHAs in CAAM that implement encryption algorithms are permanently disabled.

In encryption-disabled SoCs the following bit fields will return 0:

- CHA Number register, AESNUM bit field
- CHA Number register, DESNUM bit field
- CHA Number register, ARC4NUM bit field
- CHA Number register, PKNUM bit field

When these CHAs are disabled CAAM acts as if these CHAs are not implemented. A descriptor that attempts to use one of these CHAs will terminate with error code: Source=2h (CCB), ERRID=Fh (Invalid CHA).

Note that the following CHAs will continue to operate even in encryption-disabled SoCs:

- MDHA
- RNG

10.4 CAAM modes of operation

CAAM can operate in the following security modes:

- Trusted
- Secure
- Non-secure
- Fail

These modes are based on the current platform security state. The primary difference between these modes is that they make 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).

10.4.1 Platform Security State

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

10.4.1.1 The effect of security state on volatile keys

CAAM implements three 256-bit volatile cryptographic keys. At each power up, boot code must test and instantiate the RNG. After instantiation, (or as part of RNG instantiation), the three volatile secret keys must be generated. These values are stored within secure key registers in CAAM. The values are zeroized when CAAM transitions to fail mode (in other words, when the platform security state machine transitions to fail state).

The available volatile keys, (which are located in CAAM's secure key module), 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[1], 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 will be 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. The only paths from trusted state or secure state to non-secure state pass through fail state or through a hardware reset, and in each case the keys will be cleared. The only path from non-secure state to either trusted state or secure state is through a hardware reset, which clears the keys. Consequently, when operating in non-secure mode CAAM does not have access to trusted mode/secure mode values of these keys.

1. The JDKEK, TDKEK, and TDSK are readable and writable while in non-secure mode to facilitate hardware testing.

10.4.1.2 The effect of security state on non-volatile keys

Data that must be retained when the system is powered off must be stored in external 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. It is recommended that the master key value be unique for each device.

When CAAM is operating in trusted mode or secure mode, CAAM derives general memory blob key encryption keys (GM BKEKs) andSecure 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, a hardwired constant (all 0's) used for known-answer testing.

10.4.2 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.

10.4.2.1 Keys available in trusted mode

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

Table 10-1. Special keys used in trusted mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	• At POR, a new value (shared with secure mode but not shared with non-secure mode) should be generated from the RNG after instantiation • Zeroized when entering fail mode	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
Master key derivation key	Non-volatile, shared with secure mode, but uses a different key derivation function to generate keys not shared with trusted mode, non-secure mode or fail mode	Used for blob encapsulation or decapsulation operations

10.4.2.2 Keys available in secure mode

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

Table 10-2. Special keys used in secure mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	• At POR a new value (shared with trusted mode but not shared with non-secure mode) should be generated from the RNG after instantiation • Zeroized when entering fail mode	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
Master key derivation key	Non-volatile, shared with trusted mode, but uses a different key derivation function input to generate keys not shared with trusted mode, non-secure mode or fail mode	Used for blob encapsulation or decapsulation operations

10.4.2.3 Keys available in non-secure mode

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

Table 10-3. Special keys used in non-secure mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	• At POR, a new value (not shared with trusted mode or secure mode) should be generated from the RNG after instantiation • Zeroized when entering fail mode	• Can be read and overwritten for testing • Used for automatic key encryption and decryption when executing Job Descriptors, trusted descriptors, and shared descriptors
Trusted descriptor key encryption key		• Can be read and overwritten for testing • Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by a trusted descriptors
Trusted descriptor signing key		• Can be read and overwritten for testing • Used for testing the signing, verifying and re-signing of trusted descriptors
Master key derivation key	Non-volatile, fixed, and not shared with trusted mode or secure mode	Used for testing blob encapsulation or decapsulation operations

10.4.2.4 Keys available in fail mode

When CAAM transitions to fail mode, CAAM clears all registers that could potentially hold sensitive data[2]. Because of this, cryptographic operations that are in progress when the transition occurs will likely not produce the correct result. If this is the case, the operation completes with an error indication.

10.5 CAAM hardware functional description

As shown in Figure 10-1, CAAM functionality is aligned with several major subcomponents. This table describes these subcomponents.

Table 10-4. CAAM subcomponents

Description	Cross-reference(s)
Interfaces	
Register interface Used for access to configuration, control, status and debugging registers	Register interface (IP bus)
Job execution interface	
Job Ring Interface (JR)	Job Ring interface
Job Queue Controller	
Schedules tasks for the descriptor processor	Job scheduling
Descriptor processor	
Descriptor controller (DECO)	Descriptors and descriptor commands
Cryptographic control block (CCB)	Descriptor Controller (DECO) and CHA Control Block (CCB)
Cryptographic hardware accelerators (CHAs)	
Public key hardware accelerator (PKHA)	Public-key hardware accelerator (PKHA) functionality
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 Acclerator (AESA)	AES accelerator (AESA) functionality
Trust Architecture Modules	
Secure memory	Secure memory
Run-time integrity checker (RTIC)	Run-time Integrity Checker (RTIC)

Table continues on the next page...

2. The registers that are cleared include the class 1 and class 2 key registers, the class 1 and class 2 context registers, the math registers, the JDKEK, TDKEK and TDSK registers, the AFHA S-box, the PKHA E memory, the input data FIFO, the output data FIFO, and the descriptor buffer.

Table 10-4. CAAM subcomponents (continued)

Description	Cross-reference(s)
Secure key module	Black keys Blobs Trusted descriptors

10.5.1 System Bus Interfaces

CAAM is connected to a SoC-wide bus for access to CAAM registers. See Register interface (IP bus). An AXI master interface connects to the SoC bus fabric for DMA access to system memory. When CAAM includes Secure Memory, an AXI slave interface provides SoC access to that memory.

10.5.1.1 AXI master (DMA) interface

DMA access to system memory is implemented through an AXI master interface. CAAM DMA always asserts normal (AKA user) mode rather than privileged (AKA supervisor) mode, and always asserts data access rather than instruction access (i.e. fetch). CAAM DMA can be configured to assert either TrustZone SecureWorld or TrustZone NonSecureWorld for different bus transactions. CAAM DMA can also be configured to assert specified DID and ICID values for various bus transactions. For high throughput this interface utilizes a 32-bit data bus.

The AXI master interface configuration defaults are chosen to enhance performance where possible, however ideal configuration for performance is not the default and should not be assumed for any application. The DMA reads and writes data in data-bus-aligned bursts, whenever possible. The MCFG register LARGE_BURST field default value is '0' but better performance will be achieved with a value of '1'. Other notable performance enhancements include the use of read-safe, write-safe, and write-efficient transactions, which are described in the following sections.

10.5.1.1.1 DMA bursts that may read past the end of data structures

CAAM 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 ring or output ring if it would need to read past the end of the ring. However, as illustrated in the figure below, CAAM may read past the end of a descriptor or scatter/gather table (SGT) when fetching them because it does not know the length of the descriptor or SGT before

issuing the read transaction. If reading these additional words would cause an issue (e.g. an access control violation), the descriptor or scatter/gather table should be located at any address that avoids the issue.

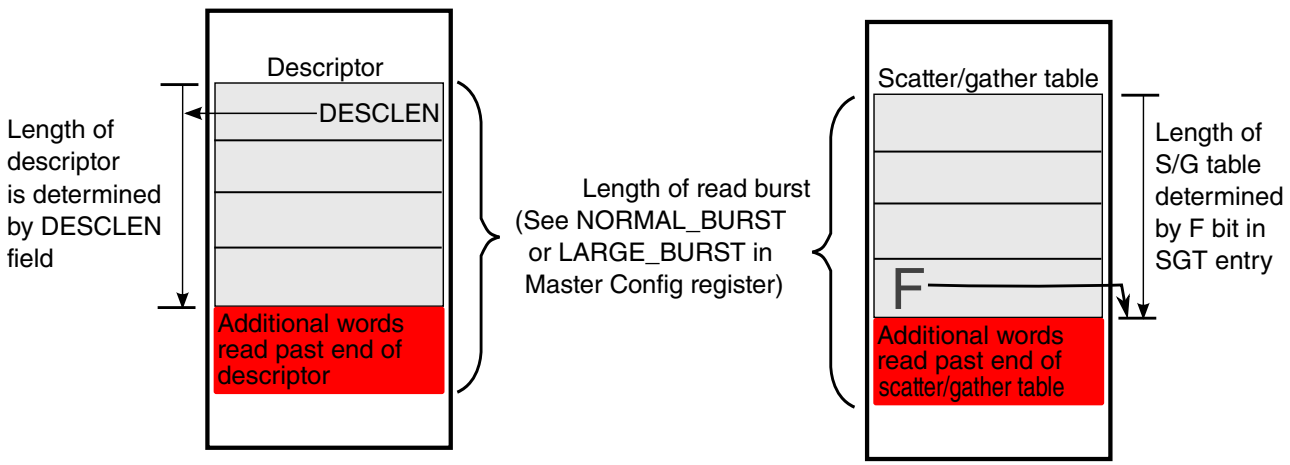


Figure 10-2. DMA may read past end of descriptor or SGT

10.5.1.2 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).

10.5.1.3 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 10-5. Summary of register interface uses

Purpose	For more information, see
During chip initialization time	
To configure CAAM, including initialization of the Job Rings.	• Initializing Job Rings
Change the default settings for CAAM's AXI DMA interface	• Master Configuration Register (MCFGR)
Configure RTIC	• Initializing RTIC
Configure Secure Memory	• Initializing Secure Memory
Initiate self-tests of CAAM's RNG	• RNG features summary
During normal steady-state operations	
Manage CAAM's Job Ring interface	• Job Ring interface

Table continues on the next page...

Table 10-5. Summary of register interface uses (continued)

Purpose	For more information, see
Alter the Secure Memory configuration	• Secure memory
During hardware and software debugging	
Read status registers and single-step descriptor commands	• Register-based service interface • CAAM Status Register • RNG TRNG Status Register • RNG DRNG Status Register • Holding Tank Status Register • Secure Memory Status Register • Job Ring Output Status Register • Job Ring Interrupt Status Register • CCB Interrupt Control Register • CCB Status and Error Register • CCB FIFO Status Register • DECO Operation Status register

NOTE

Accesses to registers other than the DECO and CCB registers must use full-word (32-bit) reads or writes. Reads and writes to the DECO and CCB registers permit byte access.

10.5.2 CAAM service interface concepts

CAAM delivers cryptographic services through a set of interfaces optimized for different use models (see Service interfaces). All service interfaces share a number of common objects and concepts, which are explained in more detail in the subsections below.

10.5.2.1 Configuring the Service Interfaces

CAAM's service interfaces must be configured before they are used. The service interfaces are configured via registers located in several 4k-byte "pages" within CAAM's register space. See the individual service interface sections in Service interfaces for configuration details. Processor access to these register pages is controlled by configuring CAAM's internal access control hardware via registers located in CAAM register page 0 (addresses 0 .. 4095). This internal access control hardware controls access to individual register pages using bus signals that indicate the DID value associated with the register bus transaction. (See Job Ring a DID Register - least significant half (JR0DID_LS - JR2DID_LS), RTIC DID Register for Block a (RTICA_DID - RTICD_DID), DECO Request Source Register (DECORSR).) A CAAM register page is accessible if the DID associated with the register read or write matches the DID in the appropriate register within CAAM register page 0.

CAAM also controls access to register pages based on the TrustZone bus signal. This signal is usually called "nonsecure" or "ns". A 1 on the ns signal indicates a NonSecureWorld bus transaction, and a 0 on the ns signal indicates a SecureWorld transaction. SecureWorld is a higher-privileged mode than NonSecureWorld, so bus transactions with ns=0 are higher privileged than ns=1 transactions. A CAAM register page can be made accessible only to TrustZone SecureWorld by setting the page's access control register appropriately.If a CAAM register page is configured for access by the TrustZone NonSecureWorld associated with a particular DID value, it can also be accessed via a SecureWorld bus transaction (ns=0) with that particular DID value, but not by SecureWorld or NonSecureWorld bus transactions with some other DID value.

10.5.2.2 CAAM descriptors

CAAM provides cryptographic services by executing a series of commands specified in CAAM descriptors. Each CAAM descriptor is formed from CAAM commands and embedded data. The set of available commands includes conditional branches, loops, subroutine calls, or jumps to other descriptors, as well as mathematical, cryptographic and data move operations. Except as specified by the branch, call, and header commands, the commands within CAAM descriptors normally execute in sequence until the descriptor has completed (or has been aborted due to external management action). Descriptors cannot change their own execution priority, but CAAM descriptors do have mechanisms to ensure coherency of data shared between descriptors.

Users of SoCs without peripheral or other direct memory access management control, such as this one, must rely on operating system or other trusted software to prevent the use of CAAM descriptors to perform unintended or malicious access to sensitive memory regions.

CAAM implements different types of descriptors to address specific processing needs:

- Job Descriptor (JD) (see Job Descriptors)

  Every CAAM job is defined by at least one JD. The JD may be provided by the CAAM service user directly via the register-based service interface or via the Job Ring-based service interface, or the JD may be created internally within CAAM in response to a service request from the Run-Time Integrity Checker (RTIC). It is also possible for a JD to invoke a Shared Descriptor (SD) or to jump to another JD, which allows a job to consist of an arbitrarily large number of commands and data objects.

- Shared Descriptor (SD) (see Shared descriptors)

SDs provide a mechanism to group and reuse instructions and data that are common in the processing of more than one related job, e.g., processing protocol data units of a network connection. Using SDs may also increase performance by improving the probability of finding an SD within CAAM that has already been read for a preceding job requiring the same processing.

- Trusted Descriptor (TD) (see Trusted descriptors)

  TDs are essentially the same as JDs, but they are cryptographically signed. When a TD is presented for execution, CAAM first checks the signature and executes the TD only if the signature is correct. TDs are intended to ensure that special access privileges are usable only by descriptors that are known to employ those privileges properly. TDs would be created by trusted software (such as secure boot software or a hypervisor), and then cryptographically signed to ensure that they were not altered by untrusted software.

- Inline Job Descriptor (IJD) (see Using in-line descriptors)

  IJDs are simply JDs that are made available to CAAM through the input data stream. JDs submitted via Job Rings may direct CAAM to execute commands from an IJD using the SEQ IN PTR command with the INL option selected.

- Replacement Job Descriptor (RJD and CRJD) (see Using replacement job descriptors)
  RJDs and Control RJDs (CRJDs) are intended to support job processing variations or updates of immediate or state data defined in SDs. Both kinds of RJDs replace the JD that invoked them and can be executed either before or after the execution of the SD. Thus RJDs and CRJDs provide the capability to permanently update or temporarily change the processing defined by SDs. RJDs may be supplied using either of two methods:
  - The normal RJD is supplied inline (like the IJD) embedded in the input data stream.
  - Alternatively, a CRJD associated with a specific SD may be utilized.

  A CRJD must be located in memory immediately following the SD. The execution of an RJD is initiated with the SEQ IN PTR command by setting the RJD control bit. For a CRJD the CTRL bit must also be set.

10.5.2.3 Job termination status/error codes

CAAM reports the termination status of all jobs, allowing software to determine whether the descriptor completed normally, with warnings, or with an error. The reporting mechanism always involves writing a job termination status word to memory. Depending on the selected service interface, CAAM may also update service interface status registers.

An all-zero status indicates that the job completed without warnings or errors. If a warning or an error was encountered, the code in the source field of the status word indicates which component within CAAM detected the condition. The remaining status word coding provides additional component-specific detail.

For jobs submitted through the Job Ring interface the job termination status is written to the Job Ring Output Status register and to the output ring in the word following the pointer to the completed job descriptor. The job termination status can be read from the Job Ring Output Status register, but because the termination status of each newly completed job will overwrite the previous job's termination status this mechanism is primarily intended to support debug and ring management (for executing single jobs or after the ring is halted). A selection of severe error conditions (potentially indicating malicious users or software instability) is stored together with additional error and/or access violation attributes in job-ring specific recoverable error record registers.

Table 10-6. Job termination status word

bits 31-28 Source	**bits 27-0** Source-specific error or warning codes					
0h (None)	0000000h - No errors or warnings					
bits 31-28 Source	**bit 27** (JMP)	**bit 26** Reserved	**bits 25-16** Reserved	**bits 15-8** (DESC INDEX)	**bits 7-4** (CHAID)	**bits 3-0** (ERRID)
2h (CCB)	See footnote [1]	0	0000h	The number of words from the start of the descriptor where the error was detected. In some cases this value may be off by one or more words due to timing issues.	0h - CCB 1h - AESA (all modes) 2h - DESA (DES and 3DES) 3h - AFHA (ARC4) 4h - MDHA (MD5, SHA-1, SHA-224, SHA-256) 5h - RNG 8h - PKHA (all public key operations)	0h - No error 1h - Mode error 2h - Data size error 3h - Key size error 3h - (RNG) Instantiate error 4h - (RNG) Not instantiated error 4h - (PKHA) A size error 5h - (RNG) Test instantiate error 5h - (PKHA) B size error 6h - (RNG) Prediction resistance error

Table continues on the next page...

Table 10-6. Job termination status word (continued)

						6h - Data out-of-sequence error 6h - (PKHA) "c" is zero for ECC F2M 7h - (RNG) Prediction resistance & test request error 7h - (PKHA) Divide by 0 error 8h - (PKHA) Modulus even error 9h - (DES) key parity error 9h - (RNG) Secure Key Generation error Ah - ICV check failed Bh - Hardware error Ch - (AES) CCM AAD size error Ch - (RNG) Continuous check error Dh - (CCB) Class 1 CHA or Class 2 CHA is not reset, or, a second CHA of the same class is selected prior to resetting the first selection Eh - (CCB) Invalid CHA combination selected Fh - (CCB) Invalid CHA
bits 31-28 Source	**bit 27** (JMP)	**bit 26** Reserved	**bits 25-16** Reserved	**bits 15-8** (DESC INDEX)	**bits 7-0** User-defined value	
3h (Jump Halt User Status)	See footnote [1]	0	0000h	The number of words from the start of the descriptor where the JUMP HALT Command was encountered.	The value in the LOCAL OFFSET field of the JUMP command is written into these bits of the termination status word. The user is free to assign any interpretation to these bits, such as using them to distinguish among different instances of the JUMP command.	
bits 31-28 Source	**bit 27** (JMP)	**bit 26** Reserved	**bits 25-16** Reserved	**bits 15-8** (DESC INDEX)	**bits 7-0** Error Code	
4h (DECO)	See footnote [1]	0	0000h	The number of words from the start of the descriptor where the error was detected. In some cases this value may be off by one or more words due to timing issues.	00h - No error 01h - SGT length error (The descriptor is trying to read more data than is contained in the SGT table.) 02h - Unused SGT entry error (Extension bit set in unused SGT entry.) 03h - Job Ring Control Error (There is a bad value in the Job Ring Control register.)	

Table continues on the next page...

Table 10-6. Job termination status word

					04h - Invalid Descriptor Command 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 non-local JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a TD to a JD, or because the target descriptor contains an SD) 0Eh - Invalid MATH Command 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. This error may result if a built-in PROTOCOL OPERATION Command has encountered a malformed PDU.) 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 into an illegal state.) 15h: Context Register Length Error. The descriptor is trying to read or write past the end of the Context Register. A SEQ LOAD or SEQ STORE with the VLF bit set was executed with too large a length in the variable length register (VSOL for SEQ STORE or VSIL for SEQ LOAD). 16h - DMA Error 1Ah - Job failed due to Job Ring reset 1Bh - Job failed due to transition to Fail Mode 1Ch - DECO Watchdog timer timeout error 1Fh - DID mismatch error (DECO was trying to share from itself or from another DECO but the two Non-SEQ DID values didn't match or the "shared from" DECO's Descriptor required that the SEQ DID and TZ/SDID values be the same but they aren't.) 20h - DECO has completed a reset initiated via the DRR register

Table continues on the next page...

Table 10-6. Job termination status word (continued)

					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.) 22h - Leading meta data is too large for rewind operation 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 - JDKEK, TDKEK or TDSK was needed, but value has not yet been initialized. 26h - A job has DECO select value for a different DECO 32h - Invalid PKCURVE Command 33h - Burster buffer reuse error (address or length went negative) 80h - DNR (do not run) error (A Job Descriptor or Shared Descriptor had the DNR bit set.) 81h - undefined protocol command 82h - invalid setting in PDB 83h - Anti-replay LATE error 84h - Anti-replay REPLAY error 85h - Sequence number overflow 86h - Invalid signature 87h - DSA Sign Illegal test descriptor 88h - Protocol Format Error (A protocol has seen an error in the format of data received. When running RSA, this means that formatting with random padding was used, and did not follow the form: 00h, 02h, 8-to-N bytes of non-zero pad, 00h, F data.) 89h - Protocol size error 8Ah - Key not written before start of protocol 8Ch - RFKG P & Q upper 100 bits the same 8Dh - RFKG computed D too small 8Eh - RFKG PDB and computed N sizes differ C1h - Undefined Blob mode C2h - Secure Memory Blob mode error C4h - Black Blob key or input size error C5h - Invalid key destination in blob command C8h - Trusted/Secure mode error in blob command CCh - Manufacturing Protection Fuse-Resident Key ECC Check error
bits 31-28	**bits 27-12**			**bits 11-8**	**bits 7-0**
Source	Reserved			NADDR	Error code

Table continues on the next page...

Table 10-6. Job termination status word (continued)

6h (Job Ring)	0000h			Number of descriptor addresses requested for error code 1Eh, otherwise 0h	00h - No error 1Eh - Error reading the Descriptor address 1Fh - Error reading the Descriptor
bits 31-28 Source	**bit 27** (JMP)	**bit 26** Reserved	**bits 25-16** Reserved	**bits 15-8** (DESC INDEX)	**bits 7-0** (COND)
7h (Jump Halt with Condition Codes)	See footnote [1]	0	0000h	The number of words from the start of the descriptor where the JUMP HALT Command was encountered.	PKHA/Math condition codes field from JUMP HALT Command.

1. If JMP = 1, the descriptor made a jump to another descriptor. When this bit is 1, the DESC INDEX field will contain the index into the descriptor that was jumped to rather than into the original descriptor.

10.5.2.4 Frames and flows

CAAM borrows the term 'frame' from network protocols. A frame simply refers to some number of sequential bytes that are usually, but not necessarily, part of a segmented byte stream and delimited by implicit or explicit start and end markers. Explicit markers are formed by so-called protocol headers and/or trailers of protocol-specific length including a possible length of 0. Implicit markers are out-of-band information defining where frame data starts and ends. For CAAM the meaning of a frame is generalized to also include designated space into which CAAM-generated frame data can be stored, as well as data that is completely unrelated to networking protocols, e.g., a piece of program code that needs to be cryptographically signed or a sequence of CAAM-generated random data.

While frames define a logical sequence of bytes, the frame data itself does not need to be necessarily stored in a single, contiguous region of memory (also referred to as a buffer). Segmented, multi-buffer frames can be formed by utilizing scatter/gather tables (stored in additional buffers) where table entries are used to keep track of frame segment address, offset, length, and other segment attributes. For details see Scatter/gather tables (SGTs).

A CAAM flow simply refers to a sequence of two or more frames requiring the same kind of processing. Whether the frame data is stored in single buffer frames, multi-buffer frames, or a mix of both is irrelevant. The key criteria of a flow is that all frames are processed in the same fashion.

Some flows require that the frames be processed in order, for instance, if there are frame data or processing context dependencies between the frames. One way to ensure that frames can be processed in a specific order is to process those frames using job descriptors submitted through the same job ring. Note that in order to improve performance CAAM may start processing the next job descriptor before the current

descriptor's write operations have completed. If the next job descriptor reads those words from memory before they have been written by the previous descriptor, the flow's data calculations may be incorrect. CAAM's shared descriptors provide mechanisms to address data dependencies within flows. For details see Shared descriptors.

10.5.2.5 User data access control and isolation

CAAM supports user data access control and CAAM-service user isolation through its ability to use service interface-specific Domain Identifiers (DIDs) and Isolation Context Identifiers (ICIDs) to tag all related memory transactions.

CAAM can be configured to assign different tags when performing associated memory accesses. Control data refers to any CAAM instructions or data utilized to process an input data stream in order to produce an output data stream or to just output status (e.g., an indication that the signature of some input data is correct). Depending on the selected service interface and user application, access to CAAM-utilized memory is managed by up to two DIDs.

When accessing memory, the RTIC service interface utilizes one DID per memory block, while each Job Ring and the register service interface may utilize up to two different DIDs and two different ICIDs configured by a trusted management entity in interface-specific CAAM registers.

10.5.3 Service interfaces

CAAM services may be invoked via the following types of service interfaces:

- A Register-based service interface
- A Job Ring interface
- A Run-time Integrity Checker (RTIC)

The register-based interface is primarily intended for management entities to use for simple one-off jobs during startup, run-time testing of CAAM functionality, or debugging CAAM descriptors. It is not intended for repetitive or high throughput activities.

The Job Ring interface provides single user/driver job queuing, job completion interrupt services, and support for dynamic service interface virtualization via a software management entity. CAAM implements 3 Job Ring interfaces that can be independently assigned (and re-assigned) to different users. This service interface type is intended to be (at least temporarily) assigned to either the Arm TrustZone (TZ), system management entities or application entities.

Jobs can also be internally initiated by CAAM's RTIC submodule. RTIC is typically configured at startup (and optionally reconfigured thereafter) by a trusted or privileged management entity, and then operates autonomously, periodically submitting specialized CAAM descriptors to the Job Queue Controller.

10.5.3.1 Job Ring interface

The Job Ring interface is a software job programming interface. For each job submitted to CAAM, software must create a job descriptor that explicitly describes the data to be processed and the keys and context to be used for the processing (see Job Descriptors).

CAAM implements 3 Job Ring interfaces. Each Job Ring interface provides an input ring for Job Descriptors and an output ring for results. This queuing mechanism allows software to schedule multiple CAAM jobs for execution and then retrieve the results as convenient. The input rings and output rings are implemented as circular buffers (also called rings) that may be located in memory external to CAAM 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. Each entry in an output ring consists of a pointer to a job descriptor followed by a job termination status word. Each output ring entry 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.

10.5.3.1.1 Configuring and managing the input/output rings, overview

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

Table 10-7. 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 Removed 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

Table continues on the next page...

Table 10-7. Input/output ring registers (continued)

Register	Description
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
Job Ring Configuration Register	Used to configure Job Ring interrupts, set endianness overrides, and select whether the optional sequence out length word appears in Output Ring entries

Figure 10-3 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 input 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 input 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 input 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.

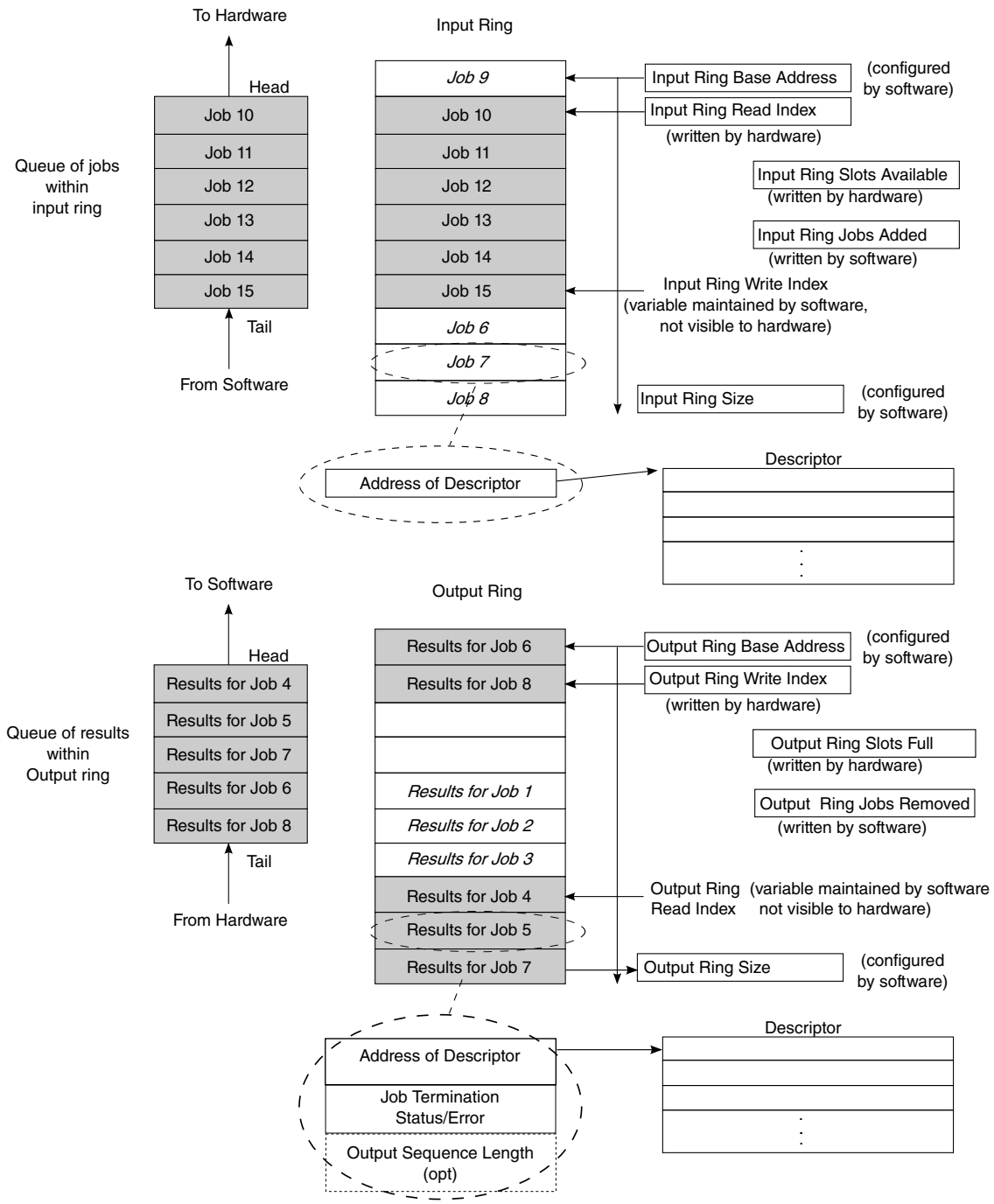


Figure 10-3. Input and output ring example

10.5.3.1.2 Managing the input rings

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

- Writes descriptor addresses starting with the next available slot into the input ring.
- Writes the number of new jobs to the Input Ring Jobs Added Register (IRJAR)

An 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 ring 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 (IRSAR) upon pulling descriptor addresses out of the ring. When software writes a new value to IRJAR, CAAM decrements IRSAR by the value that was written by software. CAAM maintains a read index in the Input Ring Read Index Register (IRRIR) that CAAM increments as it reads jobs from the input ring.

10.5.3.1.3 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, CAAM increments the Output Ring Slots Full Register (ORSFR), 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, software writes the number of jobs removed to the Output Ring Jobs Removed Register (ORJRR). CAAM decrements the output ring slots full value by the new value that software wrote to ORJRR.

Note that each entry in the output ring consists of a job descriptor address and a job termination status 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 Register (ORWIR) that CAAM increments as it places completed jobs and status into the output ring. Software can read ORWIR 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 perform a Job Ring reset via the Job Ring Command Register RESET field (JRCR[RESET]), a CAAM software reset via the Master Configuration Register SWRST field (MCFGR[SWRST]) or a power on reset. If a Job Ring reset is performed, it will clear all registers for that particular Job Ring except the IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers. The IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers can be reprogrammed or not, as appropriate, after a Job Ring reset.

10.5.3.1.4 Controlling access to Job Rings

Access to a Job Ring can be restricted to a single bus master by configuring CAAM's Job Ring DID 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 register 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 DID 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.

10.5.3.1.5 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 (with the possible exception of debugging with test data in use) also require specifying the DID values associated with the Job Ring (see the JRaDID register). These values should be configured by a trusted SoC / ring management entity. The Job Rings can also be configured for endianness and for whether to include the optional sequence out length word in the Output Ring entries (see Job Ring Configuration Register (JRCFGR)).

10.5.3.1.6 Job Ring Registers

If the Job Ring is allocated to TrustZone SecureWorld, the Job Ring registers associated with this ring can be written only via a TrustZone secure bus transaction. Nonsecure writes to Job Ring registers owned by Trustzone SecureWorld will be ignored.

When virtualization is enabled (VIRT_EN=1 in the Security Configuration register), the Job Ring registers in pages 1...3 can be written only if the corresponding Job Ring has been "started", that is, the JRSTARTR[Start_JR] bit for that Job Ring is 1. Conversely, the Job Ring configuration registers in CAAM register page 0 (for example, the JRaDID

register) can be written only if the Start_JR bit for that Job Ring is 0. The Job Ring registers are reset when the Job Ring is stopped and virtualization is enabled, in order to prepare the Job Ring for a new owner. The input ring slots available, input ring read index, output ring slots full and output ring write index registers are read-only when virtualization is disabled. These registers are writable when virtualization is enabled.

For the Job Ring register descriptions, see IRBAR and the following register descriptions.

10.5.3.1.7 Asserting Job Ring interrupts

To notify the driver software that job results are available from the Job Ring, each Job Ring interface asserts an interrupt request on a shared interrupt request line. 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.

10.5.3.2 Register-based service interface

It is possible to use the register interface to perform multiple cryptographic operations. 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.[3] This method bypasses all job scheduling performed by the Job Queue Controller. Software can perform CHA operations by writing and reading registers in the CCB directly, without using a Job Ring to run descriptors. When descriptors or commands are executed in this mode software can examine the content of most DECO and CCB registers after each descriptor or descriptor command completes. This can assist with debugging hardware and descriptor programs.

3. Note that trusted descriptors cannot be executed via the register-based service interface.

To execute descriptors or commands in this fashion software must request direct use of a DECO by writing into the DECO REQ register.[4] But before requesting a DECO, software must specify the DID and SDID values that will be used when executing descriptors under direct software control. These values are specified by selecting a Job Ring whose registers will supply the DID and SDID values. A Job Ring source is selected via the DECO Request Source register.

To use the register-based job service interface, the DECO must be programmed in proper order so that a descriptor runs correctly. The steps are:

1. Specify the DID and SDID values as described above.
2. 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 set to 1, it waits for the corresponding DECO block to complete any pending tasks before setting the DENn bit.
3. Wait for the DENn bit in the DECO Request Register to be set to 1. The Job Queue Controller sets the DENn bit to 1 when the DECO block becomes available. When this bit is set, software can use the DECO/CCB block by submitting descriptors by means of CAAM's register interface.
4. 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.
5. 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 (see the next step) is not set or if the descriptor attempts to do a STORE of type 41h to write back part, or all, of the descriptor to memory.
6. 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. If the WHL bit in the DECO Job Queue Control Register 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. SHR_FROM is not used in this format and will not be checked.
7. Wait until the DECO is done. To determine whether DECO is done, read the VALID and DECO_STATE fields in the DECOa Debug DECO register. While the job is running, VALID will be 1 and DECO_STATE will change values as the descriptor is processed. If DECO_STATE is Dh, then an error occurred. Read other fields and registers to determine the cause of the error. Note that VALID will likely remain

4. DECOs in use via the register-based service interface are not available for processing jobs from the other service interfaces. Since this version of CAAM has only one DECO, use of the register-based service interface prevents starting job descriptor processing from the Job Rings.

asserted in the event of an error. If DECO_STATE is 0h and VALID is 0, then the job finished normally.
8. Read registers of interest.
9. Done or start over.
10. 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.

Note that there are restrictions imposed when executing a descriptor under software control:

- The special cryptographic keys used to encrypt or decrypt Black Keys are not available, so Black Keys cannot be used.
- The master cryptographic key used to encrypt or decrypt Blobs is not available, so Blobs cannot be used.
- Sharing of Shared Descriptors is not permitted.
- Trusted Descriptors are not allowed.
- When virtualization is enabled, a Job Ring source must be selected in the DECO Request Source register before executing any job under direct software control. All jobs running under direct software control will then utilize the DID and SDID values for the Job Ring selected in the DECO Request Source register. When virtualization is disabled, any job under direct software control will utilize the DID and SDID values specified in the DECO DID register, and the SRC field in the Job Queue Control Register must be programmed to indicate the job is running on behalf of one of the Job Rings. Jobs cannot be executed under direct software control if those jobs appear to be from other possible CAAM sources, such as:
  - RTIC

The normal use case for the register based service interface is to debug descriptors. When such a descriptor is run through the interface and the descriptor encounters an error, once analysis of the error is done, the user can recover by releasing the DECO or by writing a 1 to the STEP bit in the DECO Job Queue Control Register. The second method allows another descriptor to be loaded and run as described above.

10.5.4 Job scheduling

The Job Queue Controller is the job scheduler within CAAM. The Job Queue Controller pulls jobs to be sent to the holding tank in round-robin fashion from the Job Rings and then from RTIC.

10.5.4.1 Job scheduling algorithm

Each time that the Job Ring's turn comes up in rotation and there is one or more jobs available in the selected Job Ring's input queue and the Job Queue Controller has internal buffer space, up to 4 jobs are fetched from the ring. Because CAAM buffers input ring entries for efficiency, several 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.

Note that RTIC requests at most one job at a time.

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

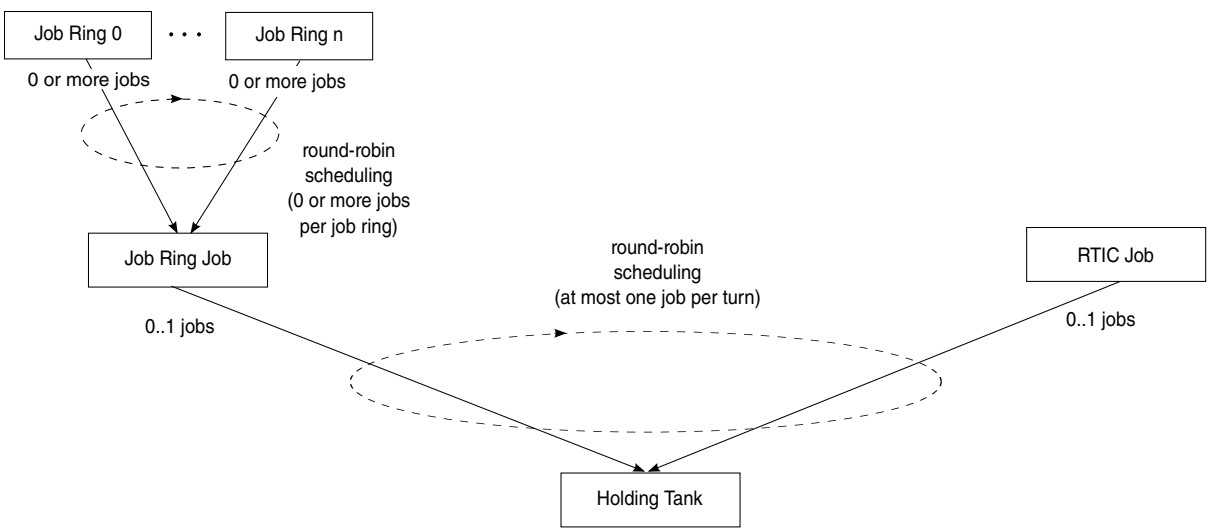


Figure 10-4. Selecting job for available holding tank

The Job Queue Controller prefetches some or all of the selected job descriptor and places it 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.

The job source (RTIC, Job Ring) is not considered when deciding which holding tank job should be assigned to the available DECO.

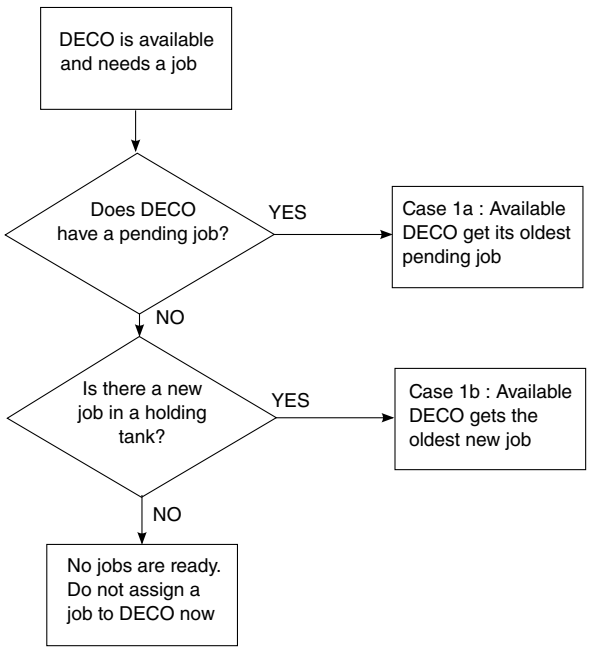


Figure 10-5. Job Queue Controller's job scheduling algorithm

10.5.4.2 Job scheduling - DECO-specific jobs

Some other versions of CAAM have more than one DECO so it is possible to specify that a job should be run in a specific DECO by using the job header extension word. This can be used to support hardware testing. Once a DECO-specific job enters a holding tank, it remains there until the specified DECO becomes available, with the following exception. If the DECO-specific job contains a shared descriptor, specifies SERIAL sharing, and the shared descriptor currently resides in a DECO other than the specified DECO, the DECO-specific job runs serially in the DECO that already contains the shared descriptor, resulting in a DECO-select error job termination code.

10.5.5 Job execution hardware

The following modules in CAAM are used to execute cryptographic or calculation acceleration jobs:

- Descriptor controller/CHA control block
- Cryptographic hardware accelerators (CHAs)

10.5.5.1 Descriptor Controller (DECO) and CHA Control Block (CCB)

The Descriptor Controller (DECO) is responsible for executing CAAM Job Descriptors. After the Job Descriptor and any Shared Descriptor referenced by that Job Descriptor are loaded, DECO begins job processing. The DECO has a dedicated CHA Control Block (CCB) that is used to control one or more 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 CHA(s). As data is produced by the CHA(s), DECO activates the DMA to write the CHA results to locations specified by the descriptor and, upon job descriptor completion, passes the completion status to the service interface originating the job request for any service-specific post processing (e.g., status report formatting). When the job processing finally completes, either successfully or with errors, the service interface takes appropriate action to inform the user.

The CCB contains all the hardware necessary to control the CHA(s) 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 DECO/CCB includes the Input data FIFO (IFIFO), Output data FIFO (OFIFO), iNformation FIFO (NFIFO), mode registers, context registers, key registers, descriptor buffer, math registers, scatter/gather tables, alignment blocks and interconnects. DECO/CCB uses all of this hardware to process Job Descriptors.

10.5.5.1.1 Alignment blocks

CAAM's internal data pathways and CHAs 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, the 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. All alignment blocks have the inputs shown in the figure, and the CCB DMA is a consumer from all alignment blocks. The Class 1 alignment block contains a nibble shift register,

which allows the Class 1 alignment block to handle data that needs to be shifted by a nibble. The only other difference between the alignment blocks is the additional consumer (Class 1 CHA, Class 2 CHA, or DECO).

The entry pulled from the iNformation FIFO 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. The alignment blocks normally transfer eight bytes of data at a time to the consumer. When the amount of data needed by the consumer is not a multiple of eight bytes, a "flush" flag or "last" flag is required to transfer the last one to seven bytes from the alignment block to the consumer.

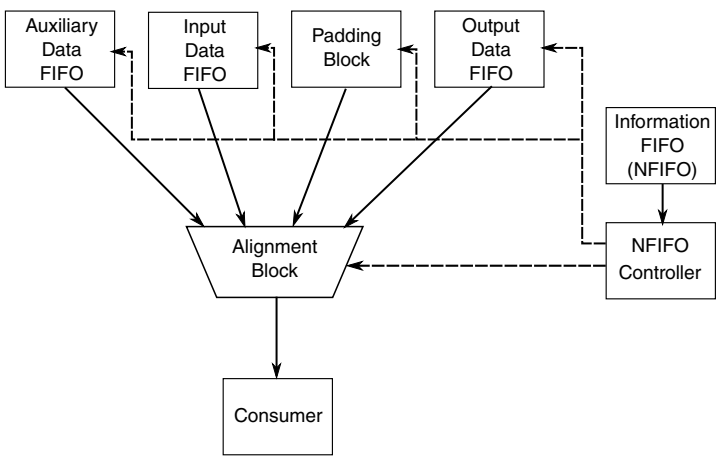


Figure 10-6. Alignment block interconnections

The Class 1, Class 2 and DECO alignment blocks can serve as the source for a MOVE command (see SRC values 8h, 9h and Ah). The DECO alignment block can also be used as a data source for a MATH command (see SRC1 field). 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. 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.

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. This info FIFO entry may be automatically or manually generated. In order to use data stored in the input FIFO, that data must be passed through one of the alignment blocks. The only other way to take data out of the input FIFO is by resetting the input FIFO, which also resets the alignment blocks.

10.5.5.2 Cryptographic hardware accelerators (CHAs) (overview)

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

- PKHA (Public key hardware accelerator)
- DESA (DES accelerator)
- AESA (AES hardware accelerator)
- MDHA (Message digest hardware accelerator)
- RNG (Random number generator)
- AFHA (ARC four hardware accelerator)

10.6 Descriptors and descriptor commands

Software's primary interaction with CAAM is through the submission of descriptors, which are small programs executed by CAAM. It is 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, or full security protocol encapsulation of a packet.

Descriptors consist of commands that are executed in sequence, although conditional and unconditional jumps are available to alter the sequence. The size of a single descriptor is limited to 64 32-bit words, but 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 usually done when the processing of a data block is dependent on the result of processing of the prior data block. Such dependencies exist for information such as sequence numbers, counter values, and cryptographic state. The current value of this sort of data can be passed from one job to another job in the same flow by embedding the data into the descriptor. This can be done using a LOAD IMMEDIATE command, or a PROTOCOL DATA BLOCK, or by embedding data words at the end of the descriptor, or by simply skipping over embedded data words. Write backs are performed using descriptor commands. Hardware does not make independent decisions regarding the fields that should be written back.

Note that to correctly use sharing flows (WAIT or SERIAL) in CAAM, if one job in the flow updates the PDB (Protocol Data Block) in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, CAAM will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

10.6.1 Job Descriptors

A Job Descriptor (JD) is a control structure that causes CAAM to execute a single job, consisting of one or more CAAM commands.

Given a pointer to a Job Descriptor, the Job Queue Controller normally will fetch from that address to the next memory bus burst boundary.[5] However, if this would be less than the number of bytes required to load an entire, maximum size, Job HEADER command, then the Job Queue Controller will fetch instead the number of bytes in the maximum size, Job HEADER command. The maximum Job HEADER size is 12 bytes.

Note that a single request is issued to read the Job Header command. If the Job HEADER command is stored across a memory bus burst boundary this request may be split into two requests under some conditions. Once all the words that comprise the Job HEADER command are received, the Job Queue Controller makes a decision. If there is a DECO available into which this job may be placed, the job is placed into the DECO for execution. If no DECO is available, or if the job can't be placed into the DECO due to sharing constraints, the Job Queue Controller will fetch the rest of the Job Descriptor if the previous reads did not already accomplish this. In addition, if there is a shared descriptor, the Job Queue Controller will also fetch the shared descriptor (unless it can be shared and is already present for another job). Once these reads have completed, the job will be eligible for placement into a DECO for execution. By prefetching all of this material, the Job Queue Controller saves the DECO from taking the time to do so, thereby signficantly improving performance.

Software-generated Job Descriptors contain pointers to the data to be operated on, and the lengths of that data. The descriptors may either directly embed security keys and context or explicitly point to these keys and context. Keys and context may also be referenced indirectly by pointing to a shared descriptor (SD) that either contains keys and context, or includes pointers to keys and context. A Job Descriptor can include a shared descriptor by reference, but a shared descriptor cannot include a Job Descriptor.

5. The first read is at least to the next burst boundary even though the descriptor may not be that long. It is up to the user to ensure that reading beyond the end of the Job Descriptor to the burst boundary will not result in any memory access errors.

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:

- Commands that specify the inputs (such as keys, IV, or data) to an operation and where to place them
- Commands that specify where to place the output(s) of the operation
- One or more OPERATION commands that specify the cryptographic work to be done

The Job Descriptor may also 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 Job 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). (see Command execution order)

Because the length of the Job Descriptor is contained in the Job HEADER command, no special termination command is required. When execution reaches the command that extends to the end of the Job Descriptor, DECO knows that the execution of the Job Descriptor has completed. Note that this endpoint is marked and does not change unless a new descriptor is loaded. Therefore, even if new descriptor material is loaded over the original material via MOVE or LOAD commands, the endpoint will not change and DECO will end execution of the Job Descriptor there. An error will be generated if DECO detects that the endpoint is inside a command. (For example, an error will be generated if the endpoint is between the words of a 2-word command.)

10.6.2 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 Black keys)

- Access to trusted descriptor-only blobs (See Blobs)
- 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. Note that each Trusted Descriptor is associated with a particular SDID, and will run only if it is executed with the same SDID as the Job Ring in which the Trusted Descriptor was created. (The signature over the Trusted Descriptor will not validate if the SDID is different.) The Trusted Descriptor can be run in the Job Ring in which it was created, or another Job Ring, as long as the SDID is correct.

Any descriptor can jump to a trusted descriptor via any of the various means: CRJD, RJD, nonlocal JUMP, or inline descriptor. However, while a trusted descriptor may use any of those means to jump, the target of such a jump must be another trusted descriptor. Otherwise, an error will be generated.

NOTE

In order to use the derived key protocol (DKP) in a trusted descriptor, the input and output for the protocol must both be via the sequence pointers. That is, the option selected must be from SEQ IN PTR to SEQ OUT PTR. There are no restrictions for other protocols.

10.6.3 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 keys, context data, and other information within the descriptor rather than referencing them with pointers. To save overhead, CAAM supports a shared descriptor mechanism. Once a shared descriptor has been fetched it is held internally for a while so it can be used by several different related jobs. The information stored within the shared descriptor 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. 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 can be amortized over many individual jobs.

In order to optimize performance when a job descriptor references a shared descriptor, use the following guidelines. The job descriptor should contain only commands specific to one job in the sequence of jobs for which the shared descriptor will be used. Such commands include where to find the input data and where to place the output data. In addition, occasional tasks such as executing an RJD or overriding the normal operation of the shared descriptor would also be found here. The shared descriptor should contain all of the generic, flow-specific, commands. That is, references to keys, context, state, operations, etc.

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. A shared descriptor always starts with a shared descriptor HEADER.

The following restrictions are specific to shared descriptors:

- A shared descriptor cannot have its own shared descriptor.
- A shared descriptor can be, at most, 62 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 10-8), 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 HEADER 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.

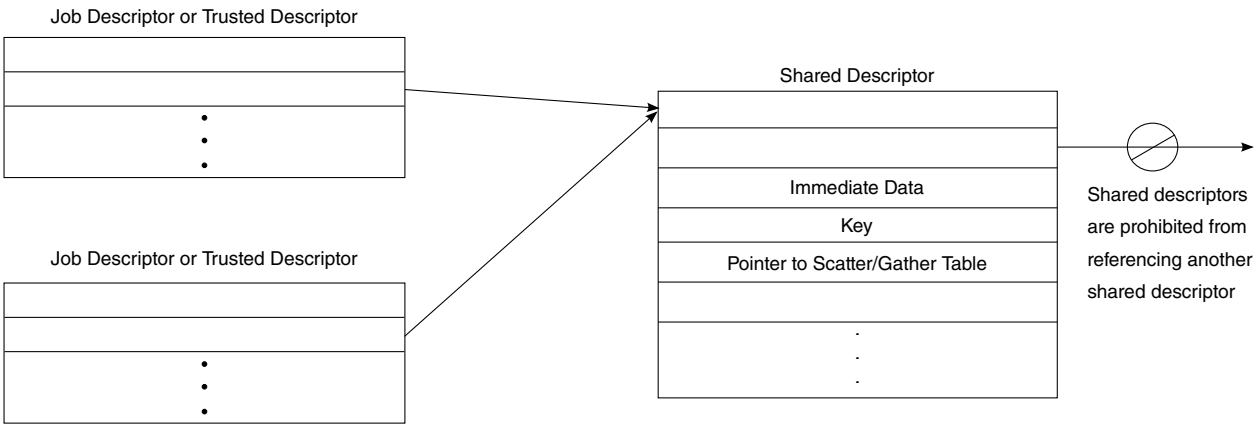


Figure 10-7. Two descriptors referencing the same shared descriptor

10.6.3.1 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. Note that to correctly use sharing flows (wait or serial), if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, CAAM will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

NOTE

If a NEVER share shared descriptor is modified during execution, and that modification is not written back to memory, the modification will NOT be seen by any other job that uses that shared descriptor. If a NEVER share shared descriptor is modified during execution, and if memory is updated with that change, subsequent jobs which reference that shared descriptor might have already fetched the original version or, if fetched during the update, might have a corrupted version of the shared descriptor. Therefore, it is up to the user to ensure that no jobs which use a NEVER share shared descriptor are in flight when the shared descriptor is updated. Clearly, NEVER share shared descriptors are not meant to be updated.

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 job descriptor header. Note that setting the REO bit in the job descriptor header tells CAAM to execute the Job Descriptor before the shared descriptor.

The sharing type may be changed to NEVER via a write to the DECO Control Register. Doing so prevents the descriptor from being shared from the DECO. The descriptor could be shared following a subsequent read from memory or from another DECO if that DECO has already gotten a copy of the descriptor. If the descriptor is being shared at the time the DECO Control Register is written to set the sharing type to NEVER, the descriptor will be shared.

Since there is only one DECO in this design, sharing can only take place by sharing the shared descriptor with the next job to execute in that DECO. If that next job isn't using the same shared descriptor, then sharing will not take place.

10.6.3.2 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, i.e. the DECO uses the same shared descriptor to process several jobs in a row without refetching the shared descriptor. This can happen with "WAIT", "SERIAL", and "ALWAYS" sharing.

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

To share shared descriptors, the SHARE bits in the job descriptor header, and sometimes in 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 10-8. 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.

Table continues on the next page...

Table 10-8. Interpretation of the SHARE fields (continued)

SHARE Name	Job Descriptor SHARE (binary)	Shared Descriptor SHARE (binary)	Description
WAIT	001	00	Share the shared descriptor once set up has completed and processing has begun. Sharing can begin after a LOAD Command (or a PROTOCOL OPERATION) 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. If the last output from a shared job is not a PDB writeback and the next shared job in the flow does not arrive in time for context to be shared, it is possible for the next job to start before the last writes from the previous job have completed. If there is a data dependency between these last writes and the reads for the subsequent job, it is possible for the reads to return the wrong (older) data. This is unlikely because data within a flow is normally shared via PDB writebacks, and because writes typically complete more rapidly than reads. But the situation can be avoided by doing a PDB update of the first few bytes of the Shared Descriptor, where "few bytes" is chosen to be bus-friendly.Class 1 and Class 2 Key Registers are shared, if valid. Context may optionally be 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			

10.6.3.2.1 Error sharing

Error sharing between jobs is not possible in single DECO designs.

10.6.3.3 Changing shared descriptors

The best shared descriptors are independent, meaning that they do not need to be modified by software for each Job Descriptor with which they are used. (Note that this is a different topic than shared descriptors that update themselves.) Shared descriptors are more easily used the more generic they are. However, shared descriptors may have to be changed on occasion, e.g., when there is a key change. Replacement job descriptors can be used for such changes to avoid requiring software to make the change.

10.6.4 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. (For more information about the SEQ IN PTR command, see SEQ vs non-SEQ commands and 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, if one is present, and job descriptor, as needed. This means the shared descriptor will no longer be executable by this job and will no longer be shareable. Note that an in-line descriptor may be scattered by means of an SGT.

Once the inline descriptor has been loaded, the detection of an error will result in a bit in the completion status being set that indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For Job Ring jobs, the original job descriptor address is placed in the appropriate output ring.

Once an inline descriptor has been loaded, use of the STORE convenience source for updating the job descriptor (41h) will result in an error.

In some cases, an inline descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

10.6.5 Using replacement job descriptors

A replacement job descriptor (RJD) 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 without having to interrupt the flow of packets.

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.

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 (that is, 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 may be executed before the MOVE completes. This could result in the intended updates not being used. If there is a chance that this may occur, use the WC bit in the MOVE command to ensure correct operation. See MOVE, MOVEB, MOVEDW, 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.

Replacement job descriptors can be trusted descriptors, and they must be trusted if the current descriptor is a trusted descriptor.

CAAM also implements a different type of RJD known as a Control RJD (CRJD). In the case of a CRJD, CAAM fetches the replacement job descriptor from memory immediately following the shared descriptor. Note that it is an error to use a CRJD for a Job Ring job if there is no shared descriptor.

Once an RJD has been loaded, use of the STORE convenience source for updating the job descriptor (41h) will result in an error.

The detection of an error will result in a bit in the completion status being set that indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For Job Ring jobs, the original job descriptor address is placed in the appropriate output ring.

In some cases, an RJD descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

Due to features of the AES encryption algorithm, special handling may be required when using a replacement job descriptor to update a key in a shared descriptor. AES encryption requires each block of data to be processed in a series of cryptographic rounds, and the AES key is successively modified at each round. When decrypting, the AES CHA must start with the fully modified form of the key (also called a decryption key or decap key) and reverse the modifications at each round, eventually ending up with the original encryption key. If the descriptor was shared, AES will have left the decryption key in the key register. However, when using a replacement job descriptor to update a key in a shared descriptor, the updated key is usually an encryption key (also called an encap key). To resolve this problem, use the LOAD command to clear the fact that the shared descriptor was, in fact, shared. That way, AES will expect the encryption key and will automatically generate the decryption key. This avoids having to generate the decryption key as part of the RJD.

10.6.6 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 scatter/gather tables. An SGT consists of one or more SGT entries. The final entry in the SGT is marked by setting the F (Final) bit in an SGT entry.

Each of the SGT entries occupies four 32-bit words, as seen in Table 10-9. 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 following table shows the SGT entry format.

Table 10-9. Scatter/gather table entry format

word 0	Reserved [32 bits] (must be 0)			
word 1	Address Pointer [LS 32 bits]			
word 2	E [1b]	F [1b]	Length [30 bits]	
word 3	Reserved [19 bits] (must be 0)			Offset [13 bits]

Table 10-10. 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 to 1, this is the last entry in the scatter/gather table.
E	Extension bit. If set to 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.
Address Pointer	Pointer to memory buffer or discontinuous scatter/gather table entry, depending on the E bit.
Reserved	Field not currently defined. Leave these bits 0 to ensure forward compatibility.

10.6.7 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 in drivers or other reference software. Some sequences of commands or combinations of command options may produce unexpected results.

10.6.7.1 Command execution order

NOTE

In the following discussion, the term Job Descriptor should be taken to include both ordinary 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'sHEADER 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.

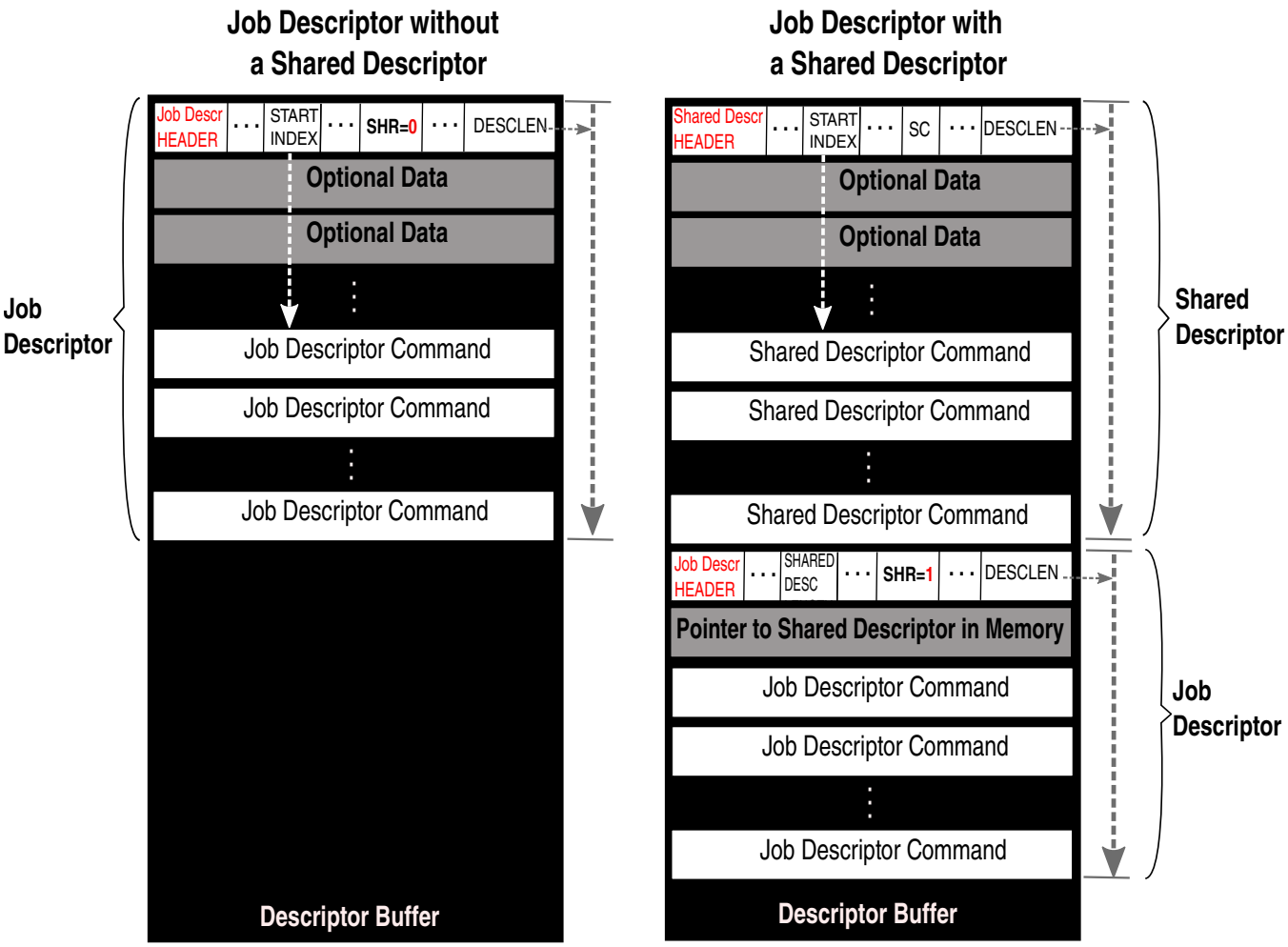


Figure 10-8. Job Descriptor layout in descriptor buffer

10.6.7.1.1 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, 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. Note that within a protocol Job Descriptor, the START INDEX value is used to skip over the PDB, if any. Before the Job Descriptor continues execution, the remainder of the Job Descriptor is fetched from memory and loaded into the DECO's descriptor buffer. The left half of Figure 10-8 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 when the JUMP is taken (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 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 should be within the current job descriptor. If the target is beyond the end of the current Job Descriptor, it is up to the programmer to ensure there is executable code at the target and that the descriptor will be able to terminate properly. One common method for proper termination is to use a JUMP HALT command.
  - 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.

10.6.7.1.2 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. The right side of Figure 10-8 shows the layout of a job descriptor that references a shared descriptor.

A pointer to the shared descriptor's location in memory appears in the word immediately following the job descriptor HEADER. Note that the job descriptor HEADER may occupy two words in addition to the shared descriptor address (see EXT field in HEADER command). The pointer, together with the DID 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. A START INDEX of 0 means start with the command immediately following the shared descriptor HEADER. If either the shared or Job Descriptor contain a PROTOCOL OPERATION command, the START INDEX value in the shared descriptor is used to skip over the PDB, if any. Note that when a shared descriptor is present, the PDB is always in the shared descriptor even if the PROTOCOL OPERATION is in the job descriptor.

10.6.7.1.3 Executing commands when REO = 0

If the Job Descriptor references a shared descriptor, the REO bit in the HEADER command determines the next command to be executed. [6] When REO = 0, DECO executes the shared descriptor before the remainder of the job descriptor, as illustrated on the left of Figure 10-9. 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. Once the shared descriptor starts executing, any job descriptor HEADER command will be treated as a no-op until a new Job Descriptor is loaded.

6. Note that the REO bit cannot be set in a trusted descriptor.

The shared descriptor ceases executing when any of the following occurs:

- an in-line job descriptor is executed
- a replacement job descriptor is executed
- a JUMP HALT command is executed
- a non-local JUMP is executed
- or the shared descriptor "falls through" to the Job Descriptor

Since the shared descriptor immediately precedes the Job Descriptor in the descriptor buffer (see right side of Figure 10-8), unless the last command of the shared descriptor causes a jump, the shared descriptor may complete by simply "falling through" to the Job Descriptor. Once the shared descriptor completes, DECO executes the Job Descriptor, starting with the Job Descriptor HEADER, which will be treated as a no-op. Execution will continue with the next command of the Job Descriptor. Execution will end following execution of the last command in the Job Descriptor unless the last command is a taken JUMP.

Once the shared descriptor HEADER command has been executed, any further shared descriptor HEADER commands will be used as absolute, unconditional, jump commands if the START INDEX field is nonzero. The START INDEX field will be used to determine the target. Note that, unlike the JUMP command, the START INDEX is the value of the target index, not a relative index. No other fields in the shared descriptor HEADER command will cause an action to take place although error conditions may be triggered. If a subsequent execution of a shared descriptor HEADER command is done where the START INDEX is zero, then the shared descriptor HEADER command will be treated as a no-op.

It is important to note the difference in how subsequent Job Descriptor HEADER commands are handled when REO=0 and REO=1.

10.6.7.1.4 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 10-9. In this case, the job descriptor command (if any) that immediately follows the shared descriptor pointer (1.1 in the diagram), or the extended HEADER word if it is present, executes immediately after the job descriptor HEADER. After the Job Descriptor completes, DECO then executes the shared descriptor commands, starting with the shared descriptor HEADER (2.0 in the diagram).

Execution of a subsequent job descriptor HEADER, other than one reached via a non-local JUMP, an RJD or inline descriptor, will terminate execution normally. Upon execution of the command which ended the Job Descriptor, no matter how many times this occurs, with the exception of taken JUMPs, execution will continue with the

command at the start of the descriptor buffer. After the first execution of the shared descriptor HEADER, subsequent executions can be used as absolute, unconditional, jumps in the same manner as subsequent shared descriptor HEADER commands are used when REO=0.

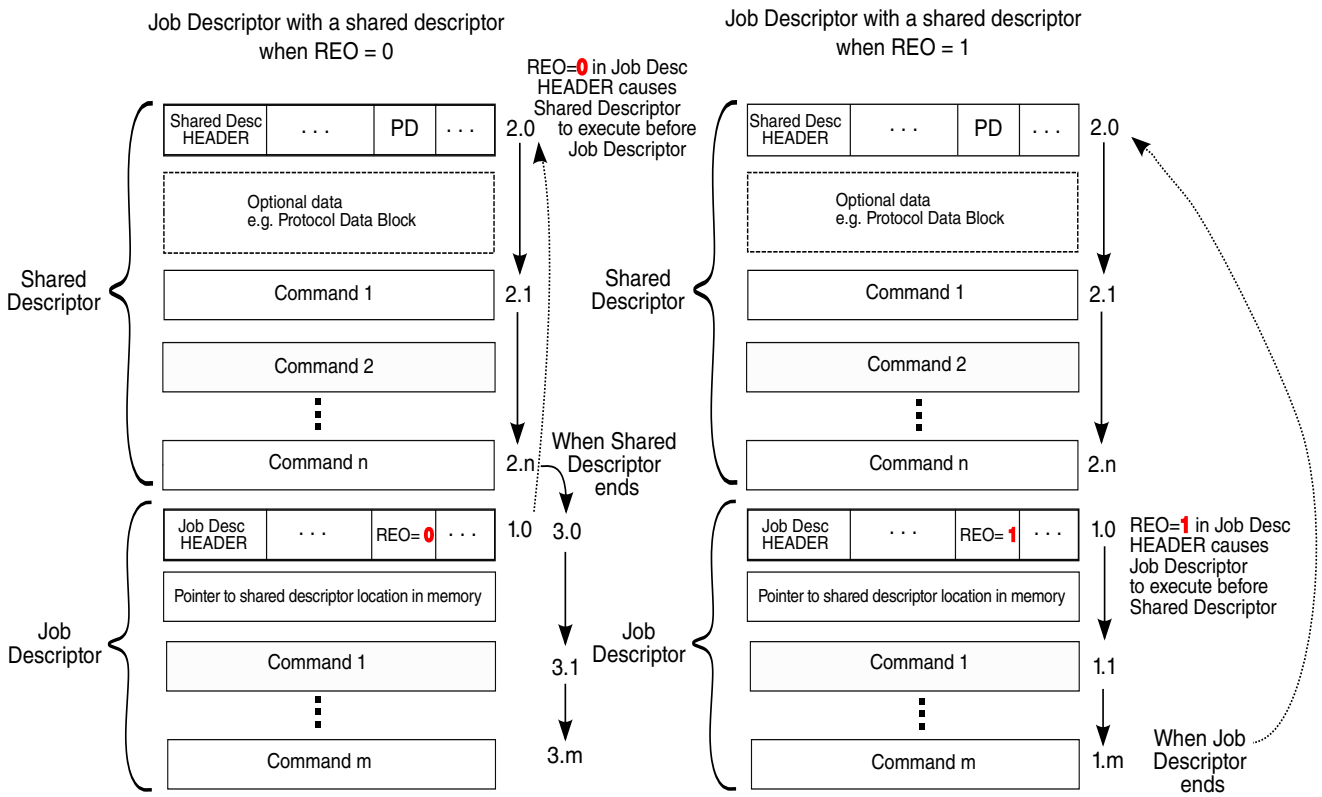


Figure 10-9. Order of command execution if a shared descriptor is referenced

10.6.7.1.5 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. 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 after the Job Descriptor runs (that is, REO = 1), but the Job Descriptor header acts as a no-op if the shared descriptor runs before the Job Descriptor runs (that is, REO = 0).

10.6.7.1.6 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 returned. 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. If an error is detected following a jump to another descriptor, a bit in the status indicates that a non-local jump was taken. Note that there is no indication of how many non-local jumps were made.

10.6.7.2 Command properties

As explained in the following subsections, three properties determine how CAAM handles each command:

- Blocking
- Load/store checkpoint
- Done checkpoint

10.6.7.2.1 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.

Many commands are blocking commands. The notable exceptions are commands that perform LOADs, STOREs, MOVEs and OPERATION algorithm commands. (That is, not PROTOCOL OPERATIONS or PKHA OPERATIONS, which are blocking.) Note that setting the WC bit in any of the MOVE commands causes the command to become blocking.

10.6.7.2.2 Load/store checkpoint

If a command is a load/store checkpoint, it must wait for certain prior LOADs and/or STOREs to complete before it can start. This property ensures that LOADs, STOREs, and other commands occur in proper order.

10.6.7.2.3 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 done once their 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. Note that done checkpoints can be for only Class 1, or only Class 2, or both Class 1 and Class 2.

10.6.7.3 Command types

The following is a list of commands that are supported in CAAM along with their blocking and checkpoint properties.

Table 10-11. List of command types

Command name	CTYPE	Blocking	Load/store checkpoint	Done checkpoint
KEY (& SEQ KEY)	00000 (00001)	Yes, if not immediate or if encrypted	Load/Store, if not immediate or if encrypted	Yes
LOAD (& SEQ LOAD)	00010 (00011)	No	Load, for some destinations	No

Table continues on the next page...

Table 10-11. List of command types (continued)

Command name	CTYPE	Blocking	Load/store checkpoint	Done checkpoint
FIFO LOAD (& SEQ FIFO LOAD)	00100 (00101)	No	Load, if immediate and another FIFO LOAD is pending or if not immediate and an immediate FIFO LOAD or MOVE to the input FIFO is pending	No
STORE (& SEQ STORE)	01010 (01011)	No	Store, if storing a scatter/gather table and that table is still being loaded	If from a Context Register the corresponding CHA must be done
FIFO STORE (& SEQ FIFO STORE)	01100 (01101)	No	Load checkpoint if encrypting	Yes if encrypting
MOVE (& MOVE_LEN) (& MOVE_B) (& MOVE_W)	01111 (01110)	Yes, if WC set	Load or Store depending on type of MOVE. It is a checkpoint if the CCB DMA is being used for a prior MOVE or for moving IMM data for KEY, LOAD, or FIFO LOAD commands.	If from a Context Register the corresponding CHA must be done
OPERATION (ALGORITHM OPERATION) (PROTOCOL OPERATION) (PKHA OPERATION)	10000	Yes, if PKHA or protocol	For PKHA	No
SIGNATURE	10010	Yes, when verifying or re-signing	Yes if recomputing signature following execution; no pending reads or writes.	No
JUMP	10100	Yes	Checkpoint based upon condition bits	If Class bit or bits are set
MATH and MATHI	10101 11101	Yes	Will wait for data if SRC1 is Input or Output Data FIFO and data is not yet available	No
Job Descriptor HEADER Shared Descriptor HEADER	10110 10111	Yes	N/A	N/A
ECPARAM	11100	No	Load if an immediate FIFO LOAD or MOVE to the input FIFO is pending	No
SEQ IN PTR	11110	Yes	Yes if there is a pending gather table read.	No
SEQ OUT PTR	11111	Yes	Yes if there is a pending scatter table read.	No

10.6.7.4 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. However, examples using the protocols that CAAM supports can be found in Protocol acceleration.) 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 nearly identical functions, with the major distinction being that the SEQ versions do not require pointers because CAAM uses sequence addresses that were defined by previously executed SEQ IN PTR or SEQ OUT PTR commands. Another difference is that SEQ commands (with the exception of SEQ STORE) do not have immediate data modes.

For jobs submitted by means of the Job Ring interface, each Job Ring can be configured via the INCL_SEQ_OUT field in each Job Ring Configuration Register to output an additional word in each entry of the Output Ring that indicates the length of the output sequence (that is, the number of bytes output via SEQ commands).

10.6.7.4.1 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 are 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 (see SEQ IN PTR command).
- A SEQ OUT PTR command to specify the length and address of the buffer for the output data (see SEQ OUT PTR command).

The SEQ IN PTR and SEQ OUT PTR commands each have an SGF field which, when set to 1, allows sequence input and/or output areas to be defined by means of scatter/gather tables.

The SEQ IN PTR command is used to create an input sequence. The SEQ OUT PTR is used to create an output sequence. 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 10-95 and Table 10-97) or by using MATHor MATHI commands to add length directly. 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 and SOP field in the SEQ IN PTR command and REW field in the SEQ OUT PTR command).

An input sequence ends when any of the following occurs:

- All specified input data is consumed (unless a rewind is then done).
- A new input sequence is started.
- An error occurs.

An output sequence ends when any of the following occurs:

- All specified output space is consumed (unless a rewind is then done).
- A new output 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. However, an input gather table can be in use by either an input sequence or by non-SEQ KEY,LOAD, or FIFO LOAD 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 DECO Gather Table Registers and DECO Scatter Table Registers).

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. (The opposite case is not a problem. For example, a non-SEQ LOAD command which

references a scatter/gather table followed by a SEQ IN PTR which references a scatter/gather table won't be an issue since the LOAD has to complete the use of its scatter/gather table before the SEQ IN PTR command can execute.)

NOTE

Hardware does not flag overwriting the scatter/gather table as an error. The descriptor programmer must ensure this does not happen.

10.6.7.4.2 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 length from packet to packet, such as packet payload. The VLF bit found in all 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 MATHor MATHI Command (see Table 10-91). This allows commands within the descriptor to calculate variable lengths. Given a total packet length, 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).

10.6.7.4.3 Transferring meta data

When processing data, CAAM typically uses the 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 (that is, 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 data to be processed, it is called 'leading meta data'. If it appears after, it is called 'trailing meta data'.

Three different tasks must be scheduled in order to transfer meta data from input to output without modification:

- Data must be read. Most often, the data is read into the input data FIFO. FIFO LOAD and SEQ FIFO LOAD are the most common methods for getting data into the input FIFO.
- Data must be stored. Most often, the data is stored from the output FIFO. FIFO STORE and SEQ FIFO STORE are the most common methods for storing data from the output FIFO.
- If the data is brought into the input data FIFO, it must be moved, via one of the MOVE commands, to the output FIFO.

While it is possible to transfer meta data without going through the input FIFO and output FIFO, such transfer methods are discouraged as timing can be complex. Furthermore, meta data is most often used with sequences so that multiple pointers don't have to be specified.

It is possible to accomplish two or three of the above tasks using a single command. This command is the SEQ FIFO STORE command with the meta data Output Data Type, 3Eh. Depending on how the auxiliary bits are set, this type of SEQ FIFO STORE will adjust the various lengths and obtain the data either from the input frame or from the input data FIFO.

The above procdures work for leading meta data. To handle trailing meta data for a sequence, start by subtracting the length of the meta data from the Sequence Input Length Register. Then, process the input frame. Once the processing is complete, add the length of the meta data back to the Sequence Input Length Register and handle the meta data. (Note that if using SEQ FIFO STORE with meta data Output Data Type, you don't need to add the length back into the register.)

10.6.7.4.4 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 and SOP field in SEQ IN PTR command and REW field in SEQ OUT PTR command. 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 data.

10.6.7.5 Information FIFO entries

The CCB has an iNformation FIFO (NFIFO). The NFIFO holds entries that describe the corresponding data to obtain from the input data FIFO, the output data FIFO, the auxiliary data FIFO, or the padding module. (The padding module provides a means for generating different types of padding and random numbers.) The data is obtained from each source in the order in which the NFIFO entries are loaded. Note that there are two

formats for NFIFO entries, one format when STYPE is not 10 and one format when STYPE is 10. Note that a single entry is able to describe the same (in-snooping) or different (out-snooping) sources for the class 1 and class 2 alignment blocks.

Typically, a command that loads data to the input data FIFO or pulls data through one of the alignment blocks will result in CAAM automatically generating the proper NFIFO entries to handle that data. However, that functionality can be overridden to allow the descriptor to directly specify the NFIFO entries. Entries can be placed into the NFIFO via a LOAD Immediate command with a DST value of 7Ah or 70h through 75h.

10.6.7.6 Output FIFO Operation

Data can be pushed into the output FIFO via:

- a LOAD IMM to the output FIFO
- a MOVE command where the destination is the output FIFO
- CHAs pushing their results into the output FIFO

It is up to the descriptor writer to ensure that there are no collisions of data from these sources. If such a collision does occur, an error will be generated.

The output FIFO does not track valid bytes. Therefore, it is up to the descriptor writer to know which bytes in the output FIFO are valid. For example, if you push 3 bytes into the output FIFO followed by 5 more bytes, these 8 bytes are not contiguous. The first three bytes are in one dword and the other 5 bytes are in a second dword. However, the output FIFO always stores 8 bytes per push. When a LOAD IMM to the output FIFO is done, the specified number of bytes are left aligned and any other bytes are written as provided. That is, if the immediate data is to be one byte, 55h, but the 4-byte value provided is 55443322h, then all four bytes are written to the output FIFO along with 4 more bytes of 0. MOVEs to the output FIFO will have similar results. However, if a CHA is pushing 3 bytes into the output FIFO, those bytes will be left aligned and the other 5 bytes will be 0.

The output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the output FIFO so each can track separately allowing consumption of data at different rates. The following list illustrates how these indices work.

- If the current NFIFO entry is not pulling data from the output FIFO, then whenever the external DMA pops an entry off the output FIFO, the two indices increment.
- If the CCB DMA pops an entry off the output FIFO, both indices will increment.
- If the DECO pops an entry off the output FIFO via the MATH command, both indices will increment.

- If the current NFIFO entry is pulling data from the output FIFO, then the two indices will track separately if the NFIFO entry is not STYPE=01 and AST=1. This is a critical point to understand: since the indices are tracking separately, if one of the consumers, either the NFIFO or the external DMA, falls far enough behind the other, the output FIFO can fill and operations will stall until the lagging consumer catches up. If the NFIFO is consuming data but there are no FIFO STOREs to advance the external DMA pointer, then the NFIFO can only consume as much data as the output FIFO can hold before a hang will result.
- If the current NFIFO entry has STYPE=01 and AST=1, the indices will both increment when the NFIFO pops entries from the output FIFO. This scenario is useful when all of the data being pushed into the output FIFO is to be consumed via one of the alignment blocks.

There are two ways to alter the behavior of the output FIFO via descriptor control. The first is the means to set the index shared by the CCB DMA, DECO, and the NFIFO to have the same value as the index used for the external DMA. This is done via a LOAD IMM to the DECO control register. The second method is to reset the output FIFO, which clears the data in the FIFO and resets both indices. The reset can also be done via a LOAD IMM to the DECO Control Register. Another means of resetting the output FIFO is via a LOAD IMM to the Clear Written register.

Another way to alter access to the output FIFO is via the OFIFO offset. This value is tracked by DECO as a means of remembering where the last access left off. For example, if a SEQ FIFO STORE of 3 bytes is done, what happens to the other 5 bytes in the output FIFO entry? Both the NFIFO entry and the FIFO STORE commands allow the descriptor to have these remaining bytes retained or discarded. If the OC bit in the NFIFO entry is set when the NFIFO is pulling data from the output FIFO, the remaining bytes are retained. A subsequent access via DECO, the CCB DMA, or the NFIFO will be able to obtain this data. However, the descriptor writer will be responsible for shifting the data as needed to get to the remaining bytes if the access is done via the DECO or the CCB DMA since only the NFIFO will be tracking where it left off.

If the CONT bit in the FIFO STORE command is set, the remaining bytes are also retained when the external DMA reads the specified number of bytes. In this case, it is DECO which tracks how many remaining bytes there are so that the subsequent FIFO STORE command will start where the prior one left off. The MOVE commands will also use the OFIFO offset, so that it can also start with the remaining data. However, the CCB DMA will always pop entries from which it takes data so that it is not possible for the CCB DMA to leave any trailing bytes in the output FIFO. Please see the section for the MOVE commands for important information on how this works.

In order to provide greater control of access to the output FIFO, the value of the OFIFO offset can be changed via a LOAD IMM to the DECO Control Register. This feature can be useful in several scenrios:

- If there are 7, or fewer, bytes of interest in the current output FIFO entry, and this data needs to be stored to memory and used within DECO but snooping is not convenient, the descriptor could do a FIFO STORE with the CONT bit set, and then do a move from the output FIFO to another destination in DECO or the CCB. (If the original OFIFO offset was nonzero, the sum of the original OFIFO offset and the number of bytes of interest must be less than 8.
- In order to do a single FIFO STORE of data that was sent to the output FIFO via separate methods, e.g. a LOAD IMM to the output FIFO followed by data from a CHA, the data must be contiguous in the output FIFO. For example, a LOAD IMM to the output FIFO of 3 bytes followed by data from the CHA would have a 5-byte gap between the loaded data and the CHA data. This could be solved by shifting the load data 5 bytes to the right and then doing a LOAD IMM of 8 bytes with those 3 bytes of interest right aligned. But now there are 5 "garbage" bytes at the start of the output FIFO data. These can be skipped over by setting the OFIFO offset to 5. Now, a single FIFO STORE can be done with a length of the number of bytes pushed by the CHA plus 3.
- The OFIFO offset can also be set to a smaller value than it currently contains. This can allow the same data to be stored twice, although the limit is back to the start of the current entry.

10.6.7.7 Cryptographic class

CAAM divides cryptographic algorithms into two different classes for the purpose of selecting CHAs. Class 1 CHAs use cryptographic keys (or, in the case of the RNG, can be used to generate keys) for encryption or decryption of data. This version of CAAM includes the following Class 1 CHAs:

- AESA, which implements the AES encryption algorithm
- DESA, which implements the DES and Triple-DES encryption algorithms
- PKHA, which implements public key cryptographic algorithms (RSA, DSA, DH, ECDSA, ECDH)
- RNG, which generates cryptographically strong random number streams

Class 2 CHAs are used for authentication of data by computing hashes (AKA message digests) of the data. This version of CAAM includes the following Class 2 CHAs:

- MDHA, which implements the MD5, SHA-1, SHA-224, SHA-256 authentication algorithms

Some key and data movement commands must have a CLASS value associated with them so they are delivered to the proper CHA.

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 10-12. Class field

Class Value	Meaning for LOAD and STORE Commands	Meaning for Other Commands
00	CCB class independent	None, sequence 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

10.6.7.8 Address pointers

Many of the descriptor commands and several data structures used by CAAM include Pointer fields. All address pointers used by this version of CAAM are 32-bits in length.

Table 10-13. 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																															

10.6.7.9 DECO/CCB behavior for jobs started via the register service interface

The DECO/CCB complex does not perform any default actions upon completion of jobs started via the register-based service interface. The service interface user is in complete control and responsible to reset all utilized registers after service use so that subsequent jobs from any service interface can be processed. Details on how to use the register-based service interface are described in the chapter Register-based service interface. A simple and effective way to clean up after use is to release the DECO, which resets the DECO/CCB complex and resets and releases any utilized CHA. Additional detail on how to handle more complex use cases is described in Using a CHA more than once in a job.

10.6.7.10 DECO/CCB default actions for one-off jobs

One-off jobs either do not utilize a Shared Descriptor (SD) or are processed with sufficiently large delays so that previously processed jobs with the same SD are no longer stored within CAAM.Any job utilizing an SD may start processing and be classified as a one-off job (not being able to share a previously processed job having used the same SD), but getting converted during job processing to be shared when CAAM becomes aware of a second job utilizing the same SD. After a one-off job completes (a potentially present SD could not be shared) all CCB/DECO registers are reset and any utilized CHAs are reset and released.

10.6.7.11 DECO/CCB actions when sharing descriptors

Sharing descriptors is only possible when CAAM becomes aware that there are two or more jobs utilizing the same Shared Descriptor (SD), i.e., this implies that two or more descriptors using the same SD have been read from memory and either are, or queued to be, processed by CAAM. Once an opportunity to share has been identified, CAAM differentiates three scenarios:

- A subsequent job using an already fetched SD can be scheduled to be processed by the DECO already processing the previous job (the DECO is self-sharing).

- A subsequent job using an already fetched SD can be scheduled to be processed by a different DECO than the DECO processing the previous job (Sharing between 2 DECOs).
- While sharing would be possible, CAAM determines that sharing would lead to unwanted preferential prioritization of a job flow sequence versus other jobs waiting for a DECO to get processed. This only occurs if the flow has been going through the same DECO. That is, one DECO has a limit as to how many jobs it can process from a particular flow if, and only if, the oldest job in the job queue waiting to be processed is not part of a flow already in a DECO. In this case sharing is suspended and the subsequent job is processed as a one-off job (see DECO/CCB default actions for one-off jobs) and thus potentially able to start another sequence of descriptor sharing.

For the former two scenarios CAAM differentiates its end-of-job processing based on the type of sharing (ALWAYS, WAIT, or SERIAL) as follows:

1. When sharing between 2 DECOs, the only items shared are the SD and, if the sharing type is not ALWAYS, the keys.
2. When self-sharing the Input FIFO, Info FIFO, and the alignment blocks, are cleared if the CIF bit in the SD HEADER is set.
3. The keys (for both Class 1 and Class 2 CHAs) are cleared for sharing type ALWAYS, but not for types WAIT or SERIAL.
4. The context (for both Class 1 and Class 2 CHAs) is cleared unless the SC bit in the SD HEADER is set.
5. The Output FIFO, any CHA done and error bits, as well as all Mode and Size registers are always cleared between jobs.

For information on sharing types see Specifying different types of shared descriptor sharing.

10.6.7.12 Using a CHA more than once in a job

Typical descriptors select CHAs, pass data through them, and then finish. However, there are times when a descriptor needs to use a CHA more than once or needs to use one CHA and then another CHA. There are several scenarios to consider. However, in all scenarios, any registers in the CCB that will get new values should be cleared. If not clearing everything in the CCB, the CCB Clear Written Register can be used to selectively clear items. For example, if the key will be used again, the CCB Key Register should be left alone, but the various size registers and context registers may need to be cleared. It is up to the descriptor writer to know which registers should be cleared and which should be left as is.

To reuse the same CHA with the same mode, clear the done flag by writing to the CCB Interrupt Control register. The CHA will start again since the CCB Mode Register is still written. (As noted above, size, context, and any other registers should be cleared prior to clearing the done flag.) Note that some CHAs may need to be reset between uses or may need other special handling. See the sections on each CHA for details.

To reuse the same CHA with a different mode, the CCB Mode Register MUST be cleared either before, or at the same time as, the CHA done bit in the CCB Interrupt Control register is cleared. The Mode Register can be cleared either using the CCB Clear Written Register or the CCB bit in the CCB CHA Control register. After this, the done flag for the CHA can be cleared via the CCB Interrupt Control register. To clear the Mode Register and the done flag simultaneously, use either the Clear Written Register selecting the CCB items, including the Mode Register, to be cleared as well as the C1RST and/or C2RST bits or use the CHA Control Register setting the CCB bit and the bit(s) of the CHA(s) to be reset. However, resetting the CHA will release a Shared CHA and arbitration will be required to reacquire it. It is up to the descriptor writer to determine if releasing the CHA between uses is desired.

To use a different CHA of the same class as a CHA that was used and is no longer required by this descriptor, use the Clear Written Register to clear the CCB Mode Register, any other CCB registers that require clearing, and the C1RST and/or C2RST bits. This will release the first CHA and allow another CHA to be selected.

NOTE

Whether a CHA is shared or dedicated, once it is selected via an OPERATION command, DECO will own that CHA until the job completes or the CHA is reset via the Clear Written Register or the CHA Control Register. Clearing the CCB Mode Register does NOT release the CHA. Since the DECO can only own one CHA of each class at any one time, to use a CHA of one class and then a different CHA of the same class, the first CHA must be reset before requesting the second CHA. If this is not done, an error will be reported and execution of that job will stop.

Note that PKHA is an exception to the above in that when a PKHA operation completes, the Mode Register and done flag are automatically cleared such that PKHA is immediately ready for another operation. However, the PKHA will still be owned and the PKHA contents and status flags will remain valid. To release the PKHA, it must be reset.

10.6.8 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 to other jobs in the same flow.

Job descriptor and Shared Descriptor HEADER commands share a base format, but some fields are specific to descriptor type. The formats of the Job Descriptor HEADER Command and Shared Descriptor HEADER Command are shown in the diagrams below, and the fields of both are described in detail below each format diagram.

Table 10-14. Job descriptor header command format

31-27				26	25	24	23	22	21-16
CTYPE = 10110b				EXT	RSD	DNR	ONE	Reserved	START INDEX / SHR DESCR LENGTH
15	**14-13**	**12**	**11**	**10-8**			**7-6**		**5-0**
ZRO	TDES	SHR	REO	SHARE			Reserved		DESCLEN
Optional additional words of HEADER command:									
If SHR = 1, a one word pointer to a Shared Descriptor is located immediately after the HEADER.									
If EXT = 1, a Job Descriptor HEADER extension is located immediately after the HEADER or, if SHR = 1, immediately after the Shared Descriptor pointer. (see Job Descriptor HEADER extension format, below)									

Table 10-15. Job descriptor header field descriptions

Job descriptor header fields	Description
31-27 CTYPE	Command type 10110b Job Descriptor HEADER
26 EXT	Extended Job Descriptor HEADER If EXT=0 : There is no extended HEADER word. If EXT=1 : The HEADER command contains an extended HEADER word, as illustrated in the format diagram below. Note that if there is no Shared Descriptor a HEADER error (13h) is generated if EXT=1 and START INDEX=1.
25 RSD	Requires DID to be the same. This bit is used to ensure that DID-based access control cannot be bypassed by sharing Shared Descriptors. If RSD=**0** the DIDs of two Job Descriptors *do not* have to match in order for them to share the same Shared Descriptor. If RSD=**1** the DIDs of two Job Descriptors *do* have to match in order for them to share the same Shared Descriptor.
24 DNR	Do Not Run If DNR=0 : Normal execution of the Job Descriptor. If DNR=1 : Do Not Run the Job Descriptor. There was a problem upstream so this descriptor should not be executed. This allows the job to be passed through the hardware and software pipeline to a point where the problem might be corrected by software and the job resubmitted. **NOTE:** 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

Table continues on the next page...

Table 10-15. Job descriptor header field descriptions (continued)

Job descriptor header fields	Description
	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.
23 ONE	One 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.
22	Reserved
21-16 START INDEX/ SHR DESCR LENGTH	Start Index or Shared Descriptor Length If SHR = 0, this is the START INDEX field Start Index specifies the position of the word in the descriptor buffer where execution of the Job Descriptor should continue following execution of the Job Descriptor HEADER. That is, DECO should jump to the specified word to continue processing. Note that if there is a HEADER extension word (EXT=1) the START INDEX must not be 1, else a 13h (header) error will result. If SHR = 1, this is SHR DESCR LENGTH field The Shared Descriptor Length specifies the length of the Shared Descriptor (in 32-bit words).
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 chips with both big-endian and little-endian processors.
14-13 TDES	Trusted Descriptor If TDES=00b : This is a normal Job Descriptor, that is, not a Trusted Descriptor. However, if the AMTD bit is set in the JRaDID register this descriptor can be run as a trusted descriptor by setting the FTD bit in the extended header word. In that case, no SIGNATURE command is required and no signature will be generated or verified. Note that if FTD=1 in the extended header word then TDES must be 00b. An error will be generated if this is not the case. If TDES=01b : This is a TrustZone SecureWorld Trusted Descriptor - a special Trusted Descriptor created by TrustZone SecureWorld. If TDES=10b : This is a TrustZone non-SecureWorld Trusted Descriptor, that is, a Trusted Descriptor created by TrustZone non-SecureWorld. If TDES=11b : This is a candidate Trusted Descriptor, that is, a descriptor that will be made into a Trusted Descriptor by appending a signature to it. If a candidate Trusted Descriptor is submitted via a Job Ring owned by TrustZone SecureWorld the descriptor will be converted to a TrustZone SecureWorld Trusted Descriptor (TDES=01b) when the descriptor is signed. If a candidate Trusted Descriptor is submitted via a Job Ring owned by TrustZone non-SecureWorld the descriptor will be converted to a TrustZone non-SecureWorld Trusted Descriptor (TDES=10b) when the descriptor is signed. Note that an error will be generated if AMTD=0 in the Job Ring's DID register. The DESCLEN field must account for the eight 32-bit words of signature which will be added. See the discussion Trusted descriptors for an explanation of Trusted Descriptors.
12 SHR	Shared Descriptor (SHR) flag If SHR=0 : This Job Descriptor does not have a Shared Descriptor and so does not include a Shared Descriptor pointer. If SHR=1 : This descriptor has a Shared Descriptor that is pointed to by the next word or words. SHR controls how START INDEX / SHR DESCR LENGTH is used.
11	Reverse Execution Order (REO). Note that this bit is ignored if SHR = 0 (that is, no Shared Descriptor).

Table continues on the next page...

Table 10-15. Job descriptor header field descriptions (continued)

Job descriptor header fields	Description
REO	If REO=0 : The Shared Descriptor is executed prior to the remainder of the Job Descriptor. If REO=1 : The Job Descriptor will be executed prior to the Shared Descriptor. Setting REO=1 in a Trusted Descriptor results in an error.
10-8 SHARE	Share State (SHARE) This defines if, and when, the Shared Descriptor referenced by this Job Descriptor can be shared with another Job Descriptor. (See Table 10-8.)
7-6	Reserved
5-0 DESCLEN	Descriptor Length 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 Job Descriptor with no Shared Descriptor.

Table 10-16. Job descriptor header extension format

31-16									
Reserved									
15-9	8	7	6	5	4	3	2	1	0
Reserved	FTD	DSELECTVALID	Reserved			Reserved			DECO_SELECT

Table 10-17. Job descriptor header extension field descriptions

Field	Description
31-9	Reserved
8 FTD	Fake Trusted Descriptor. Treat the current descriptor as a Trusted Descriptor but do not check the signature. If the descriptor is run in a Job Ring owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone SecureWorld Trusted Descriptor, otherwise the descriptor will be treated as a TrustZone non-SecureWorld Trusted Descriptor. Note that an error will be generated if FTD=1 and the source Job Ring's JRaDID register AMTD=0 (that is, the extended header says to run the descriptor as a Trusted Descriptor, but the Job Ring is not allowed to make Trusted Descriptors). In order to use FTD, the TDES field in the first word of the header command must be 00b. An error will be generated if this is not the case. If FTD=1, no SIGNATURE commands are required. If any SIGNATURE skip commands are present they will be treated as no-ops. If a final SIGNATURE command is present, it will be treated as the end of the descriptor.
7 DSELECT VALID	DECO_SELECT field is valid. If DSELECTVALID=0 : Any DECO can run the job. The DECO_SELECT field is ignored. If DSELECTVALID=1 : The job must be run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, DECO error 026h will be generated.
6-4	Reserved
3-1	Reserved
0 DECO_SELECT	DECO Select

Table 10-17. Job descriptor header extension field descriptions

Field	Description
	If DSELECTVALID = 1, the job is run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, an error will be generated. Note that for programming consistency, a one-DECO version of CAAM has a one-bit DECO_SELECT field. **NOTE:** DECO-specific jobs have the possibility to create a deadlock in CAAM when they are used as part of a flow. Therefore, it is strongly recommended that DECO-specific jobs either not be part of a flow or all the jobs in the flow be assigned to the same DECO.

Table 10-18. Shared descriptor header format

31-27					26	25	24	23	22	21-16
CTYPE = 10111					Reserved	RIF	DNR	ONE	Reserved	START INDEX
15	**14**	**13**	**12**	**11**	**10**	**9-8**		**7-6**		**5-0**
ZRO	Reserved	CIF	SC	PD	Reserved	SHARE		Reserved		DESCLEN

Table 10-19. Shared descriptor header field descriptions

Shared descriptor header fields	Description
31-27 CTYPE	Command type 10111b Shared descriptor
26	Reserved
25 RIF	Read Input Frame 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. The data is read into the input data FIFO. This is the equivalent of a SEQ FIFO LOAD of the entire input frame without an NFIFO entry or any data size registers being written. This bit is intended to allow DECO to issue reads from the SEQ IN PTR as soon as possible, thereby reducing processing latency. However, there are contraindications to its use: • If the descriptor contains any LOADor KEY command that is not immediate. • If the descriptor contains a KEY command that loads an encrypted key. The Derived Key PROTOCOL OPERATION command is not included in this restriction. • If the descriptor contains a SEQ IN PTR with RTO (Restore Input Sequence) set • If the descriptor contains a PROTOCOL OPERATION Command specifying either Blob encapsulation or Blob decapsulation. There are restrictions with the use of RIF with Public Key Cryptography operations: RIF may be used, but only if all the input FIFO is drained by other descriptor commands before the PROTOCOL COMMAND is encountered.
24 DNR	Do Not Run 0 Normal execution 1 Do Not Run. There was a problem upstream so this descriptor should not be executed. **NOTE:** 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

Table continues on the next page...

Table 10-19. Shared descriptor header field descriptions (continued)

Shared descriptor header fields	Description
	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.
23 ONE	One 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.
22	Reserved
21-16 START INDEX	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.
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 chips with both big-endian and little-endian processors.
14	Reserved
13 CIF	Clear Input FIFO (CIF) If set, the input FIFO and the NFIFO entries are reset between self-Shared Descriptors. That is, these are reset if the next job to be run within the same DECO has the same Shared Descriptor as the previous job run in that same DECO. (The Input FIFO and NFIFO are always reset between descriptors that don't share the same Shared Descriptor.)
12 SC	Save Context (SC) After this descriptor completes, if Serial-Sharing is selected, and if sharing of the Shared Descriptor occurs within the same DECO (self-sharing): 0 The context registers are cleared. 1 The context registers are maintained and used by the subsequent descriptor. Save Context is intended to allow multiple subsequent Shared Descriptors to maintain context when an operation is split across multiple jobs.
11 PD	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. And, if the DNR bit was not already set, update the Shared Descriptor in memory so that the DNR bit is set there as well.
10	Reserved
9-8 SHARE	Share State (SHARE) The SHARE field in the Shared Descriptor HEADER is used when the SHARE field in the Job HEADER is defer: 100b. This field then determines if, and when, the Shared Descriptor referenced in this Job Descriptor can be shared with another Job Descriptor. (See Table 10-8.) Also see Specifying different types of shared descriptor sharing for further information.
7-6	Reserved
5-0 DESCLEN	Descriptor Length This field represents the total length in 4-byte words of the Shared Descriptor. A Shared 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 the maximum size of a Shared Descriptor is 62 words (assuming that the Job Descriptor consists of only the Job Descriptor HEADER command and the pointer to the Shared Descriptor).

If the SHR bit in a Job Descriptor HEADER command is set, a pointer to the Shared Descriptor immediately follows the header.

10.6.9 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, or PKHA E-Memory. The SEQ KEY command is identical to the KEY command except that no address is specified and the VLF bit replaces the SGT bit and the AIDF bit replaces the IMM bit.

If the key to be loaded into a key register is encrypted, 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 PKHA E-Memory or the AFHA Sbox or key registers because only KEY commands decrypt Black Keys, and these registers are the only possible destinations for KEY 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.

If ENC is set (that is, 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

The KEY command is blocking under the following circumstances:

1. Decrypting a black key.
2. Loading a red key that is NOT immediate.
3. CHAs are not done.
4. The data must pass through the input FIFO and there are info FIFO entries in the way.
5. The data must be read into the data FIFO and there is other data in the input data FIFO that is in the way. (This does not apply to SEQ KEY AIDF.)
6. The CCB DMA is required but is busy.
7. The hardware which schedules external reads is required but is busy.

Table 10-20. KEY command format

31–27	26–25	24	23	22	21	20	19–18	17–16
CTYPE = 00000 or 00001	CLASS	SGF or VLF	IMM or AIDF	ENC	NWB	EKT	Reserved	KDEST

15	14	13	12–10	9–0
TK	PTS	PKL E	Reserved	LENGTH
Additional words of KEY command:				
Pointer (one word) or Value (if immediate, one or more words)				

Table 10-21. KEY command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=00000b : KEY command If CTYPE=00001b : SEQ KEY command
26-25 CLASS	Class. This defines whether this key is for a Class 1 or Class 2 algorithm. If CLASS=01b : Class 1 Key If CLASS=10b : Class 2 Key All other values are reserved. **NOTE:** The CLASS field must be set to Class 1 if the Key Destination (KDEST) field is set to 01b or 10b. The CLASS field must be set to Class 2 if the KDEST field is set to 11b.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF) If CTYPE = 00000b (KEY), this bit is the Scatter/Gather table Flag (SGF). If SGF=0: Pointer points to actual data. If SGF=1: Pointer points to a scatter/gather table.

Table continues on the next page...

Table 10-21. KEY command field descriptions (continued)

Field	Description
	NOTE: It is an error if both the SGF bit and the IMM bit are set. It is also an error for the SGF bit to be set when reading a key from a Secure Memory key partition (that is, a partition whose SMAP and SMAG register settings do not permit an ordinary read transaction). If CTYPE = 00001b (SEQ KEY), this bit is the Variable Length Flag (VLF). If VLF=0: The number of bytes of data to be loaded into the key register is specified by the LENGTH field. If VLF=1: The number of bytes of data to be loaded into the key register is specified by the value in the VSIL register.
23 IMM or AIDF	Immediate Flag (IMM) or Already in Input Data FIFO (AIDF) If CTYPE = 00000 (KEY), this bit is the IMM flag. If IMM=0 : The key value is found at the location pointed to by the pointer in the next word. If IMM=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. **NOTE:** PKHA E-Register values can be very large and may not fit within the descriptor buffer. An AFHA SBOX cannot be supplied as immediate data because it is too large to fit in the descriptor buffer. It is an error if both the IMM bit and the SGF bit are set. If CTYPE = 00001b (SEQ KEY), this bit is the AIDF flag. If AIDF=0 : Read the Input Data Sequence data and load it into the specified destination. If AIDF=1 : Do not read the Input Data Sequence data since it is already in the Input Data FIFO, but load the data in the Input Data FIFO into the specified destination. It is an error for ENC and AIDF to both be 1.
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, the input and output data FIFOs, Class 1 Mode register, Key Size, context, and key registers. Therefore, Class 2 Black Keys (if any) must be loaded prior to loading the Class 1 register, and Class 1 Black Keys must be loaded prior to loading any of the resources noted above.
21 NWB	No Write Back If NWB=0 : Enables write-back of key loaded in key register. Note that it is not usually possible to write a key back out to memory as plaintext, but if NWB=0 the key can be written out as a Black Key by using the FIFO STORE command. If NWB=1 : Prevents the key that is loaded into the key register from being written back out to memory. NWB applies to all available key locations: • Class 1 Key Register • Class 2 Key Register • PKHA E Memory • AFHA S-Box Setting this bit sets the key register's NWB flag. 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. It is an error if NWB is not set when reading a key from a Secure Memory key partition (meaning a partition whose SMAP register settings do not permit an ordinary read transaction).
20 EKT	Encrypted Key Type The EKT bit determines which decryption mode is used when a Black Key (ENC = 1) is loaded. If EKT=0 : The Black Key is decrypted using AES-ECB. If EKT=1 : The Black Key is decrypted using AES-CCM.

Table continues on the next page...

Table 10-21. KEY command field descriptions (continued)

Field	Description
	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 it is possible that no error will be issued but the value loaded into the key register will be incorrect. Note that an error status is generated if EKT=1 and ENC=0.
19-18 Rsrvd	Reserved. Must be 0.
17-16 KDEST	Key Destination If KDEST=00b : The key is loaded into the respective Key Register selected by the CLASS field. If KDEST=01b : The key is loaded into the PKHA E-Memory. This key destination requires CLASS = 01b (Class 1 key). In versions of CAAM through Era 10 (including this version), loading a plaintext value to PKHA E-RAM via the KEY command clears the contents of the Class 1 Key Register and the C1 Key Size Register. Starting with Era 11 this side-effect no longer occurs. If KDEST=10b : The key is loaded into the AFHA S-box. This key destination requires CLASS = 01b (Class 1 key). In versions of CAAM through Era 10 (including this version), loading a plaintext value into the AFHA S-box via the KEY command clears the contents of the Class 1 Key Register and the C1 Key Size Register. Starting with Era 11 this side-effect no longer occurs. If KDEST=11b : The key is regarded as a Derived HMAC Key, and is loaded into the Class 2 Key Register. This key destination requires CLASS = 10b (Class 2 key).
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. If TK=0 : Use the Job Descriptor Key Encryption Key (JDKEK) to decrypt the key that is to be loaded into a key register. If TK=1 : A trusted descriptor wants to use the Trusted Descriptor Key Encryption Key (TDKEK) to decrypt the key that is to be loaded into a key register.
14 PTS	Plaintext Store If PTS=0: The key loaded cannot later be stored in plaintext form. If PTS=1: The key loaded can later be stored in plaintext form using a FIFO STORE or SEQ FIFO STORE command. Note the following restrictions: The AFHA S-box can be stored in plaintext form if the S-box was loaded with a KEY command with PTS=1 or if a key is loaded with a KEY command with PTS=1 into the Class 1 Key register and the AFHA is run in INIT mode to create an S-box. The Class 2 Key register can be stored in plaintext form if a Derived HMAC Key key was loaded into it with a KEY command with PTS=1 or if a key is loaded into the Class 2 Key register with a KEY command with PTS=1 and the MDHA is run in INIT mode to create a Derived HMAC Key. An error is generated • if PTS=1 & ENC=1. • if PTS=1 & NWB=1. • if PTS=1 & KDEST=01b (PKHA E-Memory). • if the Class 1 Key register is loaded with a KEY command and the AFHA S-Box was previously loaded with a KEY command with PTS=1. • if a key is stored from the Class 2 Key register after the Class 2 Key register was loaded with a KEY command with PTS=1. • if the Class 1 Key register is stored after the AFHA S-box or the Class 1 Key register was loaded using a KEY command with PTS=1.
13	PKHA Little Endian key load

Table continues on the next page...

Table 10-21. KEY command field descriptions (continued)

Field	Description
PKLE	If PKLE=0: The key is not swapped as loaded into PKHA E-Memory. If PKLE=1: The key is swapped as loaded into PKHA E-Memory. This feature is intended to support curves like Curve25519 and Ed25519, which are typically represented as little-endian byte strings, but not for Weierstrass curves, which are typically represented as big-endian byte strings. An error is generated if PKLE=1 and KDEST is not 01b (i.e., if not loading to PKHA E-Memory). PKLE must be set to 0 for black (encrypted) keys.
12-10 Rsrvd	Reserved. Must be 0.
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.
Additional words of KEY command:	
POINTER	If IMM = 0, this field is a pointer to the key to be loaded. If IMM = 1, this field is not present. **NOTE:** This field is not present for SEQ KEY Commands.

10.6.10 LOAD commands

NOTE

In the following discussion, the term 'LOAD command' refers to both the SEQ and non-SEQ forms of the command.

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, the VLF bit replaces the SGF bit, and the immediate bit cannot be set. See SEQ vs non-SEQ commands. (Note that while SEQ KEY and SEQ FIFO LOAD have an AIDF bit, SEQ LOAD does not.)

When reading from an external address, the LOAD command, whether SEQ or non-SEQ, uses hardware within DECO to schedule DMA transactions. This command will block until that hardware is available. For LOAD IMM, if the DMA hardware is required but is in use, the command will block until the DMA hardware becomes available. (The command may block for other reasons as well, as documented in a following table.) Once

the command is handed off to the responsible hardware, descriptor execution will continue with the next command. Therefore, the requested data may not be present for some time. It is up to the descriptor writer to ensure that the data arrives prior to attempting to use it. Paying attention to the blocking nature just discussed is critical in order to avoid hanging descriptors.

The definitions of the OFFSET and LENGTH fields in the LOAD command can depend on the CLASS and destination (DST) fields. The first table below shows the LOAD command fields, the second table defines the fields, and the third table defines the legal destinations and how each destination affects the other fields.

Table 10-22. LOAD command format

31–27	26-25	24	23	22–16
CTYPE = 00010 or 00011	CLASS	SGF or VLF	IMM	DST
15-8			**7**	**6-0**
OFFSET			PKLE	LENGTH *(when a PKHA Size Register is the LOAD destination, or when the NFIFO is the destination, and DTYPE selects PKHA)*
OFFSET			LENGTH *(8 bits, not a PKHA size register or data type for destination)*	
Additional words of LOAD command:				
Pointer (one word, see Address pointers) or Value (if immediate, one or more words)				

Table 10-23. LOAD command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=00010b : LOAD command If CTYPE=00011b : SEQ LOAD command
26-25 CLASS	Class. The algorithm class of the data to be loaded. If CLASS=00b : Load class-independent objects in CCB. If CLASS=01b : Load Class 1 objects in CCB. If CLASS=10b : Load Class 2 objects in CCB. If CLASS=11b : Load objects in DECO.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF) flag. Meaning depends on CTYPE. If CTYPE = 00010 (LOAD), this bit is the Scatter/Gather table Flag (SGF). If SGF=0 : The pointer points to actual data. If SGF=1 : The pointer points to a scatter/gather table. **NOTE:** 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 Flag (VLF). If VLF=0 : The LENGTH field indicates the length of the data.

Table continues on the next page...

Table 10-23. LOAD command field descriptions (continued)

Field	Description
	If VLF=1 : The length of the data is variable. CAAM uses the length in the Variable Sequence In Length register rather than the value in the LENGTH field. However, an error will be generated if the values in the VSIL register and OFFSET field are not a valid combination as indicated in table Table 10-24.
23 IMM	Immediate Flag If CTYPE = 00010 (LOAD) If IMM=0, the data to be loaded is found at the location pointed to by the address pointer. If IMM=1, the data to be loaded follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up, if necessary, to the nearest 4-byte word. **NOTE:** If the SGF bit is set, it is an error for this bit to be set. If CTYPE = 00011 (SEQ LOAD) IMM must be set to 0. Setting IMM to 1 generates an error.
22-16 DST	The DST value defines the destination register. See Table 10-24 for a list of supported destinations.
15-8 OFFSET	OFFSET defines the start point for writing within the destination.
7 PKLE	PKHA Little Endian data load If PKLE=0: data associated with this load of a size register or NFIFO entry is not swapped as loaded into a PKHA Memory If PKLE=1: data associated with this load of a size register or NFIFO entry is swapped as loaded into a PKHA Memory. This feature is intended to support curves like Curve25519 and Ed25519, which are typically represented as little-endian byte strings, but not for Weierstrass curves, which are typically represented as big-endian byte strings.
7-0 *(for PKHA: 6-0)* LENGTH	Length of the data. The value in the DST field determines whether the length is specified in bytes or words. See Table 10-24 for details.
Additional words of LOAD command:	
31-0 POINTER/ VALUE	Address pointer if IMM = 0 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. **NOTE:** This field is present only for LOAD Commands (that is, not for SEQ LOAD Commands).

CAAM can accomplish the data transfer associated with a LOAD immediate command in two different ways:

- Using a direct (non-DMA) path to the register, referred to as a direct immediate load
- Using CAAM's internal-transfer DMA

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 transfer mechanisms (see following paragraph).
- If the data length or offset restrictions are not met, CAAM automatically selects the internal-transfer DMA data path (the second bullet above).

The direct immediate load is the most efficient of the transfer mechanisms, but it has the following restrictions:

- It can transfer only 4 or 8 bytes, unless the destination is the input Data FIFO, the Auxiliary Data FIFO, or the output Data FIFO, in which case the length must be no more than 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
  - 4 bytes with any multiple of a 4-byte offset if the destination is a context register
  - 8 bytes with any multiple of an 8-byte offset if the destination is a context register

As shown in Table 10-24, 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 10-24, 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, the following data-swapping operations can be configured:

- byte-swapping
- half-word swapping

and these swapping operations for control data registers can be configured independently from the swapping for message data registers.

The same swapping operatings can be configured independently for each Job Ring (see the Job Ring Configuration Register (JRCFGR)).

NOTE

For those destinations that allow immediate loads with a nonzero offset, the combination of offset=0 length=4 is equivalent to the combination of offset=4 length=4. This has been done to maintain backward compatibility. Both of these

combinations will load the right-most word of the destination. Therefore, in order to load the left-most word of the destination, the combination must be offset=0 length=8. This is ONLY the case when IMM=1 and, therefore, does not affect the SEQ LOAD command. When IMM=0, offset=0 length=4 will load the left-most word of the destination while offset=4 length=4 will load the right-most word. This behavior does not affect any other commands.

Table 10-24. 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	C1KSR	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				C2KSR	Class 2 Key Size Register	
02	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	C1DSR	Class 1 Data Size Register	Writes to the data size registers block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready.
	10				C2DSR	Class 2 Data Size Register	
03	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	C1ICVSR	Class 1 ICV Size Register	
	10				C2ICVSR	Class 2 ICV Size Register	
05	11	Control	See below	Yes	DCTRL2	DECO Control Register 2	See notes below.
			The DECO Control Register 2 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: LENGTH[4]: Transfer the value in the output frame tracking length register to the Variable Sequence Output Length Register. All other bits of OFFSET and LENGTH are reserved and must be 0.				
06	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	CCTRL	CHA Control Register	
	11	Control	See below	Yes	DCTRL	DECO Control Register	See notes below.

Table continues on the next page...

Table 10-24. 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
			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]: Reserved OFFSET[5:4]: Reserved OFFSET[3]: Disable Automatic NFIFO Entries (If disable and enable are both set, disable dominates) OFFSET[2]: Enable Automatic NFIFO Entries OFFSET[1:0]: Change Share Type 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				
07	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ICTRL	IRQ Control Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	DPOVRD	DECO Protocol Override Register	If bit 31 = 1 the value loaded into DPOVRD overrides the default values in some protocol PDB fields. See individual protocol sections for usage details. If bit 31 = 0, DPOVRD is not used as an override by the built-in protocols. The other bits are defined on a protocol by protocol basis. This register may be used as a source or destination by MATH and MATHI commands.
08	00	Control	4/0 bytes	Yes	CLRW	Clear Written Register	

Table continues on the next page...

Table 10-24. 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
			8/0 bytes 4/4 bytes				
	11	Control	0-32/ 0-7 bytes	No	MATH0 W	DECO Math Register 0 (Words)	1, 2
09	11	Control	0-24/ 0-7 bytes	No	MATH1 W	DECO Math Register 1 (Words)	1, 2
0A	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	CISEL	CHA Instance Select Register	-
	11	Control	0-16/ 0-7 bytes	No	MATH2 W	DECO Math Register 2 (Words)	1, 2
0B	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	AADSZ	AAD Size Register	-
	11	Control	0-8/ 0-7 bytes	No	MATH3 W	DECO Math Register 3 (Words)	1, 2
0F	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ALTDS1	Alternate Data Size Class 1 Register (aliased to the Class 1 Data Size Register)	The ALTDS1 destination can be used only with a LOAD Immediate command. Writes to the ALTDS1 block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready.
10	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKASZ	PKHA A Size Register	This holds the size of the data in the PKHA A RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
11	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKBSZ	PKHA B Size Register	This holds the size of the data in the PKHA B RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
12	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKNSZ	PKHA N Size Register	This holds the size of the data in the PKHA N RAM.

Table continues on the next page...

Table 10-24. 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
							The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
13	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKESZ	PKHA E Size Register	This holds the size of the data in the PKHA E RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
20	01	Message	0-128/ 0-128 bytes	No	CTX1	Class 1 Context Register	A LOAD IMM to a context register blocks if there are any outstanding external loads to either context register. A non_IMM LOAD to a context register blocks if the CCB DMA is writing to either context register.
	10	Message	0-128/ 0-128 bytes	No	CTX2	Class 2 Context Register	
30	11	Control	0-32/ 0 bytes	No	MATH0DW	DECO Math Register 0 (Double Word)	1, 3
31	11	Control	0-24/ 0 bytes	No	MATH1DW	DECO Math Register 1 (Double Word)	1, 3
32	11	Control	0-16/ 0 bytes	No	MATH2DW	DECO Math Register 2 (Double Word)	1, 3
33	11	Control	0-8/ 0 bytes	No	MATH3DW	DECO Math Register 3 (Double Word)	1, 3
38	11	Message	0-32/ 0-7 bytes	No	MATH0B	DECO Math Register 0 (Bytes)	1, 4
39	11	Message	0-24/ 0-7 bytes	No	MATH1B	DECO Math Register 1 (Bytes)	1, 4
3A	11	Message	0-16/ 0-7 bytes	No	MATH2B	DECO Math Register 2 (Bytes)	1, 4
3B	11	Message	0-8/ 0-7 bytes	No	MATH3B	DECO Math Register 3 (Bytes)	1, 4
40	01	Message	0-96/0-95 bytes	No	KEY1	Class 1 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 the latter case, before the value written into the Key
	10	Message	0-128/ 0-127 bytes	No	KEY2	Class 2 Key Register	

Table continues on the next page...

Table 10-24. 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
							register is used as a key the KEY SIZE register must be written by a separate command. A LOAD IMM to a Key Register blocks if there are any outstanding external loads to either Key Register. A non_IMM LOAD to a Key Register blocks if the CCB DMA is writing to either Key Register.
	11	Control	1-64/ 1-63	No	DESC BUF	DECO descriptor buffer	See comments below.
			For LOADs into the Descriptor Buffer the values in the LENGTH and OFFSET field are specified in 4-byte words. An error is generated if the sum of the LENGTH and OFFSET fields is greater than 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 descriptor buffer. 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	NFIFO and size register(s)	Using the Immediate data, write an NFIFO entry and load the Size register from the DL or PL field in that NFIFO entry. Also see note below.
-	-	-	This creates an NFIFO 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 table below titled "Which Size Registers are loaded" indicates which size registers are loaded.				
71	00	Control	0-3 0 bytes	Yes	NFSM	NFIFO and size register(s)	Using the Immediate data write an NFIFO entry and load the size register using for PL or DL 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	NFIFO	Using the Immediate data write an NFIFO entry. This is equivalent to DST value 7Ah. Also see note below.
-	-	-	This creates an NFIFO entry from 4 or 8 bytes of IMM data. No size registers are written.				
73	00	Control	0-3 0 bytes	Yes	NFM	NFIFO	Using the Immediate data write an NFIFO entry filling in the DL or PL field from the least-significant 32-bits of the MATH

Table continues on the next page...

Table 10-24. 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
							register selected via the three 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 NFIFO 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 NFIFO. The table below titled "Which Size Registers are loaded" indicates 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 three ls bits of the LENGTH field. No NFIFO entry is loaded. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.
NOTE: 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 NFIFO. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded. **NOTE:** For DST values 70h, 72h, 74h, and 7Ah, the direct destination depends on the value in the LENGTH field. If the LENGTH is 4, then the direct destination is the NFIFO. However, if the LENGTH field is 8, then the direct destination is special control hardware which breaks up large lengths so that the maximum length permitted in an NFIFO entry is not exceeded. This hardware pushes as many entries into the NFIFO as necessary. The special control hardware will stall if the NFIFO is full, resuming when space becomes available. **NOTE:** For DST values 70h-75h and 7Ah, the LOAD will block if the NFIFO is full. In addition, further access to the NFIFO will block if the hardware which breaks up large entries is in use.							
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 NFIFO entry is encountered. Thereafter, input to the Class 1 Alignment Block will not be nibble shifted unless the IDFNSR is written again. Any nibble remaining in the shift register will remain there once the last or flush is seen. This means that if there are N bytes of data, to get the last nibble out requires NFIFO entries totaling N+1 bytes.				
77	00	Message	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ODFNS	Output Data FIFO Nibble Shift Register	See notes below.

Table continues on the next page...

Table 10-24. 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
			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. Any valid nibble remaining is always pushed into the output FIFO following the assertion of CHA Done.				
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 will stall if there is no room left in the AUXDATA buffer. The AUXDATA path should be treated in the same way as the input DATA FIFO. If NFIFO entries are not used properly, execution will hang if more LOADs are done to the AUXDATA buffer but room can't be created by draining that buffer via NFIFO entries and the corresponding alignment block being drained by CHAs or the MOVE command. Note that data can also be supplied to the Auxiliary Data FIFO by using a MOVE command.				
7A	00	Control	4/0 bytes 8/0 bytes ** **	Yes	NFIFO	NFIFO	See notes below.
			The NFIFO can be written by means of the FIFO LOAD command or a MOVE command, or by using this DST value with a LOAD Immediate command. ** If LENGTH = 8, the LOAD command is interpreted as follows: word 1: LOAD IMM, LENGTH = 8, DST = 7Ah word 2: bits [31:12] contain the NFIFO entry if not padding type, else bits [31:10] contain the NFIFO entry word 3: Extended Length (DECO creates as many NFIFO entries as needed to satisfy the Extended Length - see the notes above)				
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. The data must be left-aligned and byte swapping will be done if the endianness settings require it. This LOAD will block if there is no more room in the input data FIFO or if there is other data heading to the input data FIFO. Care should be taken since this block could turn into a hang if the LOAD is unable to proceed.				
7E	00	Message	1-8/0 bytes	Yes	OFIFO	Output Data FIFO	See notes below.
			Must use a LOAD Immediate command. The data must be left-aligned and byte swapping will be done if the endianness settings require it. This LOAD will block if there is no more room in the output data FIFO. It is up to the descriptor writer to ensure that this LOAD will not collide with a move to the output FIFO or a push to the output FIFO by one of the C1 CHAs. Care should be taken since this block could turn into a hang if the LOAD is unable to proceed.				
All combinations of value and class that do not appear in this table are reserved							

1. May be affected by protocols. 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 math registers are in contiguous addresses, it is possible to load more than one math register simultaneously. A LOAD IMM to a math register blocks if there are any outstanding external loads to any math register. A non_IMM LOAD to a math register blocks if the CCB DMA is writing to any math register.
2. When this destination is used, the data loaded into the Math register will be treated as words.
3. When this destination is used, the data loaded into the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers. It is recommended that only full double words be loaded.
4. When this destination is used, the data loaded into the Math register will be treated as bytes.

The following table details which size registers are written when loading NFIFO entries when one, or more, size registers are also written. (DST values 70h, 71h, 74h, and 75h.) An entry of "None/Reserved" means that no size register will be written and NXP reserves the right to assign that DTYPE to some size register in the future. Therefore, such DTYPEs should not be used as they could break compatibility.

Which Size Registers are Loaded

DTYPE (hex)	If PKHA selected	If Class 1 CHA selected, but not PKHA	If Class 2 CHA selected
0	PKHA A Size	None/Reserved	Class 2 Data Size
1	PKHA A Size	AAD Size and Class 1 Data Size	Class 2 Data Size
2	PKHA A Size	IV Size and Class 1 Data Size	Class 2 Data Size
3	PKHA A Size	AAD Size and Class 1 Data Size	Class 2 Data Size
4	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
5	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
6	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
7	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
8	PKHA N Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
9	PKHA E Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
A	None/Reserved	ICV Size	If PKHA selected: Class 2 ICV Size If PKHA not selected: if both Class 1 and Class 2 CHAs selected, Class 2 Data Size else Class 2 ICV Size
B	None/Reserved	None/Reserved	None/Reserved
C	PKHA A Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
D	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
E	None/Reserved	None/Reserved	None/Reserved
F	None/Reserved	Class 1 Data Size	Class 2 Data Size

10.6.11 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, PKHA data (other than for the E Memory), 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, the command contains an AIDF bit in place of the IMM bit, and the command includes a VLF bit instead of a SGF bit. See SEQ vs non-SEQ commands.

Because the only destination is the input data FIFO, this command does not include a DST field. 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. 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. This command will block for a variety of reasons:

1. FIFO LOAD IMM will block if the input FIFO is full.
2. FIFO LOAD IMM will block if the DMA is required to move the data but the DMA is busy.
3. If a DMA transaction is required, the FIFO LOAD command will block if the hardware which schedules DMA transactions is in use.
4. If there are external reads destined for the input data FIFO, FIFO LOAD IMM will block until that data arrives.
5. The FIFO LOAD command uses the same logic as the LOAD command does when loading NFIFO entries with LENGTH=8. If an NFIFO entry is required and this logic is busy, the command will block.

Table 10-25. FIFO LOAD command format

31–27	26–25	24	23	22	21–16
CTYPE = 00100 or 00101	CLASS	SGF or VLF	IMM or AIDF	EXT	INPUT DATA TYPE

15	14–10	9–0
PKLE	reserved	LENGTH *when FIFO LOAD data type selects a PKHA register*
LENGTH *(16 bits, when PKHA is not selected)*		
Additional words of FIFO LOAD command:		
Pointer (one word, see Address pointers) or Value (if immediate, one or more words)		
EXT LENGTH (one word, present if EXT=1)		

Table 10-26. FIFO LOAD command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=00100b : FIFO LOAD command If CTYPE=00101b : SEQ FIFO LOAD command
26-25 CLASS	Class. Cryptographic algorithm class. If CLASS=00b : Used for SEQ FIFO LOAD only. Skips the specified length in memory without scheduling any read transactions and no data is actually read. However, Scatter/Gather Table entries will be read as needed. FIFO INPUT DATA TYPE field is ignored. No info FIFO entry is generated. If CLASS=01b : Load FIFO with data for a Class 1 alignment block. If CLASS=10b : Load FIFO with data for a Class 2 alignment block. If CLASS=11b : Load FIFO with data for both Class 1 and Class 2 alignment blocks (both In Snoop and Out Snoop; the INPUT DATA TYPE will distinguish between them). **NOTE:** The CLASS field must be non-zero for FIFO LOAD commands because the 00b 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 Flag (VLF). If CTYPE = 00100b (FIFO LOAD), this bit is the Scatter/Gather table Flag (SGF). If SGF=0, the pointer points to actual data. If SGF=1, the pointer points to a Scatter/Gather Table. **NOTE:** If the IMM bit is set, it is an error for this bit to be set. If CTYPE = 00101b (SEQ FIFO LOAD), this bit is the Variable Length Flag (VLF). If VLF=0, the LENGTH field indicates the length of the data. If VLF=1, the length is variable. CAAM uses the length in the Variable Sequence In Length register and ignores the LENGTH field. **NOTE:** It is an error to set VLF = 1 when the EXT bit = 1.
23 IMM or AIDF	Immediate Flag(IMM) or Already in Input Data FIFO (AIDF) If CTYPE = 00100 (FIFO LOAD), this bit is the Immediate Flag (IMM). If IMM=0 and SGT=0 the data begins at the location pointed to by the Pointer field, but if SGT=1 the data begins at the location pointed to by the Scatter-Gather Table, which is pointed to by the Pointer field. If IMM=1, the data follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up, if necessary, to the nearest 4-byte word. **NOTE:** It is an error if this bit is set when SGF = 1 or EXT = 1. If CTYPE = 00101 (SEQ FIFO LOAD), this is the Already in Data FIFO (AIDF) bit. If AIDF is 0, CAAM will read the input sequence data from memory. If AIDF is 1, CAAM will not read the input sequence data (because it is already in the Input Data FIFO). This form is convenient since the NFIFO and Data Size Registers will be loaded automatically if Automatic Info FIFO Entries is enabled. As a result, a 1-word command can replace two 2-word commands.
22 EXT	Extended Length If EXT=0 : Input data length is solely determined by the 16-bit LENGTH field,

Table continues on the next page...

Table 10-26. FIFO LOAD command field descriptions (continued)

Field	Description
	If EXT=1 : Input data length is determined by the 32-bit EXTENDED LENGTH. If the INPUT DATA TYPE indicates a bit length, then the EXTENDED LENGTH field contains the number of full bytes, and the right 3 bits of the LENGTH field, if nonzero, indicate the number of valid bits in the last byte. See Bit length data. **NOTE:** It is an error if this bit is set when IMM is also set.
21-16 INPUT DATA TYPE	FIFO input data type See Table 10-28 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 PKLE *(If data type selects a PKHA register)*	PKHA Little Endian data load If PKLE=0: data is not swapped as loaded into a PKHA Memory. If PKLE=1: data is swapped as loaded into a PKHA Memory. This feature is intended to support curves like Curve25519 and Ed25519, which are typically represented as little-endian byte strings, but not for Weierstrass curves, which are typically represented as big-endian byte strings. PKLE must be set to 0 for black (encrypted) keys. **NOTE:** In this instance of CAAM DECO will report an 'Invalid FIFO LOAD command' error if PKLE is set and the length is encoded in the first word. Instead set the EXT bit and encode length in the extension word, or set the VLF bit and utilize VSOL.
15-0 *(for PKHA data type: length in 9-0)* LENGTH	Length of data If EXT = 0 : LENGTH = number of bytes of input data or for bit-length message data, the number of bits of input data. Setting EXT=0, VLF=0 and PKLE=1 will cause an error. If PKLE must be set, use EXT=1 or VLF=1. If EXT = 1 : The EXT LENGTH field indicates the number of full bytes of data. If the INPUT DATA TYPE indicates a bit-length, the right 3 bits of the LENGTH field, if nonzero, indicate the number of valid bits in the last byte. See Bit length data.
Additional words of FIFO LOAD command:	
POINTER	If IMM = 0, this field is a pointer to the data to be loaded. If IMM = 1, this field is not present. **NOTE:** This field is not present for SEQ FIFO LOAD Commands.
31-0 EXT LENGTH	For EXT = 0, this field not present. For EXT = 1, EXTENDED LENGTH specifies number of full bytes of data to load. For bit-length data, the least-significant 3 bits of the LENGTH field indicate the number of valid bits in an additional byte of data. See Bit length data.

10.6.11.1 Bit length data

If the 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. Note that the entire 33rd byte is read and it is up to the consumer of that last byte to know that only the specified number of bits in the last byte are valid.

The Number of Additional Valid Bits is placed in the NUMBITS field of the Class 1 and/or Class 2 Data Size Register. The NUMBITS field is not visible to any functional logic in CAAM other than a subset of the CHAs. (The NUMBITS field may be read via SkyBlue or a store of the Data Size Registers.)

The CHAs that receive the NUMBITS field are:

- AESA, which will error if it sees a nonzero NUMBITS field.

The following CHAs do not receive the NUMBITS field:

- PKHA
- DES
- AFHA
- MDHA
- RNG

It is possible to use a nonzero NUMBITS field with a CHA which does not receive the NUMBITS field. To do this, add 1 more to the proper Data Size Register. However, note that the remaining bits in the last byte will be whatever values they were at the source of that byte. That is, the remaining bits are not masked to 0.

It is not possible for CAAM to automatically concatenate two separate bit fields. For example, if an NFIFO entry for 3 bits is followed by an entry for 5 bits, these entries will NOT result in a one-byte entry. To achieve such concatenation, use the shift operations in the MATH command.

When using automatic NFIFO entries with the FIFO LOAD command to specify bit lengths, C1 must always have Flush or Last set and C2 must always have Last set. Failure to set the Last and Flush bits as stated will result in an error. When manually generating NFIFO entries no error will be generated if the Flush or Last bits are not set as suggested. However, as stated above, each incomplete byte's remaining bits will have whatever values they had at the source.

If the FIFO LOAD command is used to generate the NFIFO entry, the number of bytes specified in the LENGTH field for the NFIFO entry will be the number of full bytes if the NUMBITS field is zero, and the number of full bytes plus one if the NUMBITS field is nonzero. This ensures that the CHA will consume the last, partial, byte. If the descriptor writer is manually generating the NFIFO entries, care must be taken to handle the length properly.

Table 10-27. Specifying data with residual bit length

15–0	
Number of Bits	
Alternate interpretation:	
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.

10.6.11.2 FIFO LOAD input data type

Table 10-28 contains an enumeration of the various built-in input data FIFO data types. This field is ignored if neither of the Class bits are set. Only message data can have a bit length (as opposed to a byte length), and such message data must always have Flush or Last set.

Note that the NFIFO source type is not needed, as it is always inferred from the Table 10-28. The length for the NFIFO entry is the amount of data being placed in the data FIFO. (One is added to the length in the case of bit-length data if the number of bits in the last byte is nonzero.) 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.

Note also that data should not be left in the Input Data FIFO with the expectation that it will be shared with a subsequent shared descriptor executing in the same DECO.

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 NFIFO entry.

Table 10-28. FIFO LOAD input data type field

FIFO Input Type Field Bit #						Meaning
21	20	19	18	17	16	
00b		PKHA				PKHA Register Load All values not specified below are reserved. This data is always flushed. An error is asserted if the length is larger than fits in the PKHA RAM.

Table continues on the next page...

Table 10-28. FIFO LOAD input data type field (continued)

FIFO Input Type Field Bit #						Meaning
21	20	19	18	17	16	
						NOTE: Loading quadrants of a given PKHA register with different-sized values may cause invalid data to be loaded into the quadrants. To avoid this issue, make sure that all quadrants of a given register have the same size values by left-filling short values with zero. If it is necessary to load different-sized values in quadrants of the same register, insert a JUMP command between quadrant loads (which will wait for automatic information FIFO entries to be processed): jump jsl = 1, type = 0, cond = nifp, local offset = 1. **NOTE:** The PKHA E RAM can not be loaded via the FIFO LOAD command using automatic NFIFO entries. Use the KEY command or get the data into the input data FIFO without an automatic NFIFO entry and then manually create an NFIFO entry and write the PKHA E Size register.
		0	0	0	0	PKHA A0
		0	0	0	1	PKHA A1
		0	0	1	0	PKHA A2
		0	0	1	1	PKHA A3
		0	1	0	0	PKHA B0
		0	1	0	1	PKHA B1
		0	1	1	0	PKHA B2
		0	1	1	1	PKHA B3
		1	0	0	0	PKHA N
		1	1	0	0	PKHA A
		1	1	0	1	PKHA B
00b		1	1	1	1	Place the data into the input Data FIFO but do not generate an NFIFO entry and do not write any size registers, even if automatic NFIFO entries are enabled.
010b			LC2	LC1	FC1	Message Data
011b			LC2	LC1	FC1	Message Data for Class 1 out-snooped to Class 2
100b			LC2	LC1	FC1	IV. No padding is done for Class2.If Last or Flush for Class 1 is set, Class 1 is padded to 16-byte boundary with 0. No padding is done for Class 1 if the data naturally ends on a 16-byte boundary.
101b			1	LC1	FC1	Bit-length message data. Last of Class 2 treated as set even if not set in the command; either LC1 or FC1 must be set
110b			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.
111b			LC2	LC1	FC1	ICV
If CLASS 1 is asserted, LC1 means the data defined in the current NFIFO entry is the last data. When LC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, LC1 is ignored. If CLASS 2 is asserted, LC2 means the data defined in the current NFIFO entry is the last data. When LC2 = 1, the last byte of the specified length is made available from the Class 2 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 2 is negated, LC2 is ignored If CLASS 1 is asserted, FC1 means the data defined in the current NFIFO entry is the last data of this type. When FC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, FC1 is ignored.						

Table continues on the next page...

Table 10-28. FIFO LOAD input data type field (continued)

FIFO Input Type Field Bit #						Meaning
21	20	19	18	17	16	
Note that the difference between LC1 and FC1 is only important when the data is going to a CHA. If the data is to be consumed by the CCB DMA, then FC1 should be used as LC1 may confuse a CHA. LC2 should not be used if the data will be consumed by the CCB DMA. In such cases, it is better to use a manual NFIFO entry with the FC2 bit set.						
All values not specified are reserved.						

10.6.12 ECPARAM command

The ECPARAM command is used to load one parameter from a set of built-in elliptic curve parameters into one of the PKHA registers. This command will block until any data already in transit to the Input Data FIFO has been received.

The DEST value 1111b is used to leave the selected parameter in the Input Data FIFO. No NFIFO entry is generated. If automatic NFIFO entries have been disabled, the ECPARAM command will leave the parameter in the Input Data FIFO and no NFIFO entry is generated. Note that when no NFIFO entry is generated, no PKHA size register is written. Furthermore, the descriptor writer is responsible for the NFIFO entry to get this parameter to its final destination.[8, 9] In all other legal cases, the ECPARAM command will result in loading the parameter into the selected PKHA register by automatically creating the NFIFO entry and automatically writing the correct PKHA size register with the correct value.

Table 10-29. ECPARAM command, format

31–27		26-20	19–16
CTYPE = 11100		0100000	DEST
15	**14-8**	**7–4**	**3–0**
Reserved	DOMAIN	Reserved	PARAMETER

Table 10-30. ECPARAM command, field descriptions

Field	Description
31-27	Command type

Table continues on the next page...

8. This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries.
9. If parameters of different sizes are to be loaded into different quadrants of the same PKHA register, then it is up to the user to ensure that the first parameter is completely loaded before executing the second ECPARAM command. This is because the same size register is used and can't be changed for the second parameter until the first has been successfully loaded.

Table 10-30. ECPARAM command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>CTYPE</td><td>IF CTYPE=11100b : ECPARAM command</td></tr>
<tr><td>26-20</td><td>0100000b. This particular 7-bit value is mandatory. Any other value will generate an error.</td></tr>
<tr><td>19-16
DEST</td><td>Destination. This field specifies which PKHA register to load:
0000b - PKHA A0
0001b - PKHA A1
0010b - PKHA A2
0011b - PKHA A3
0100b - PKHA B0
0101b - PKHA B1
0110b - PKHA B2
0111b - PKHA B3
1000b - PKHA N
1100b - PKHA A
1101b - PKHA B
1111b - Input Data FIFO--This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries.
All other values are reserved.</td></tr>
<tr><td>15</td><td>Reserved</td></tr>
<tr><td>14-8
DOMAIN</td><td>Elliptic Curve Domain. This field selects one of the built-in elliptic curve domains:
<table>
<tr><th>DOMAIN</th><th>ECC Domain Selected</th></tr>
<tr><td>0h</td><td>P-192</td></tr>
<tr><td>1h</td><td>P-224</td></tr>
<tr><td>2h</td><td>P-256</td></tr>
<tr><td>3h</td><td>P-384</td></tr>
<tr><td>4h</td><td>P-521</td></tr>
<tr><td>5h</td><td>brainpoolP160r1</td></tr>
<tr><td>6h</td><td>brainpoolP160t1</td></tr>
<tr><td>7h</td><td>brainpoolP192r1</td></tr>
<tr><td>8h</td><td>brainpoolP192t1</td></tr>
<tr><td>9h</td><td>brainpoolP224r1</td></tr>
<tr><td>Ah</td><td>brainpoolP224t1</td></tr>
<tr><td>Bh</td><td>brainpoolP256r1</td></tr>
<tr><td>Ch</td><td>brainpoolP256t1</td></tr>
<tr><td>Dh</td><td>brainpoolP320r1</td></tr>
<tr><td>Eh</td><td>brainpoolP320t1</td></tr>
<tr><td>Fh</td><td>brainpoolP384r1</td></tr>
<tr><td>10h</td><td>brainpoolP384t1</td></tr>
<tr><td>11h</td><td>brainpoolP512r1</td></tr>
</table></td></tr>
</table>

Table continues on the next page...

Table 10-30. ECPARAM command, field descriptions

Field	Description	
	12h	brainpoolP512t1
	13h	prime192v2
	14h	prime192v3
	15h	prime239v1
	16h	prime239v2
	17h	prime239v3
	18h	secp112r1
	19h	wtls8
	1Ah	wtls9
	1Bh	secp160k1
	1Ch	secp160r1
	1Dh	secp160r2
	1Eh	secp192k1
	1Fh	secp224k1
	20h	secp256k1
	40h	B-163
	41h	B-233
	42h	B-283
	43h	B-409
	44h	B-571
	45h	K-163
	46h	K-233
	47h	K-283
	48h	K-409
	49h	K-571
	4Ah	wtls1
	4Bh	sect113r1
	4Ch	c2pnb163v1
	4Dh	c2pnb163v2
	4Eh	c2pnb163v3
	4Fh	sect163r1
	50h	sect193r1
	51h	sect193r2
	52h	sect239k1
	53h	Oakley3a
	54h	Oakley4a
	All values not specified are reserved.	
	1. The "r", "R^2mod r" and "k" parameters are not valid for either OAKLEY domain. 2. The "C" parameter is not valid for Fp domains (i.e. DOMAIN < 40h).	

Table continues on the next page...

Table 10-30. ECPARAM command, field descriptions (continued)

Field	Description
7-4	Reserved
3-0 PARAMETER	Elliptic Curve Parameter. This field specifies which elliptic curve parameter is to be loaded into the PKHA register specified by DEST. PARAMETER / EC Domain Parameter Selected: 0h: q 1h: r^{-1} 2h: Gx 3h: Gy 4h: a 5h: b 6h: $R^2 \bmod q$ 7h: $R^2 \bmod r^{-1}$ 8h: c 9h: k^{-1} All values not specified are reserved. 1. The "r", "R^2mod r" and "k" parameters are not valid for either OAKLEY domain. 2. The "C" parameter is not valid for Fp domains (i.e. DOMAIN < 40h).

1. The "r", "R^2mod r" and "k" parameters are not valid for either OAKLEY domain.
2. The "C" parameter is not valid for Fp domains (i.e. DOMAIN < 40h).

10.6.13 STORE command

NOTE

In the following discussion, the term 'STORE command' refers to both the SEQ and non-SEQ forms of the 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 and the VLF bit replaces the SGF bit. 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 10-31 shows the command fields, and Table 10-33 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 stored from control data registers are regarded as word-oriented data, whereas data stored from message data registers are regarded as byte strings.

To facilitate operation in chips with different endianness configurations, the following data-swapping operations can be configured:

- byte-swapping
- half-word swapping

and these swapping operations for control data registers can be configured independently from the swapping for message data registers.

The same swapping operations can be configured independently for each Job Ring (see the Job Ring Configuration Register (JRCFGR)).

Table 10-31. STORE/SEQ STORE command format

31–27	26–25	24	23	22–16
CTYPE = 01010 or 01011	CLASS	SGF or VLF	IMM	SRC
15–8			**7–0**	
OFFSET			LENGTH	
Additional words of STORE/SEQ STORE command:				
Pointer - present for STORE, but not for SEQ STORE (one word, see Address pointers)				
If immediate (IMM = 1), one or more words of data appear here				

Table 10-32. STORE/SEQ STORE command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=01010b : STORE command If CTYPE=01011b : SEQ STORE command
26-25 CLASS	Algorithm class of the data object to be stored See the SRC field for additional explanation. If IMM = 1 a value other than 00b in the CLASS field will cause an error to be generated. If IMM = 0, the following definitions are used: If CLASS=00b : Store class-independent objects from CCB. If CLASS=01b : Store Class 1 objects from CCB. If CLASS=10b : Store Class 2 objects from CCB. If CLASS=11b : Store objects from DECO.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF). If CTYPE = 01010b (STORE), this bit is the Scatter/Gather table Flag (SGF). If SGF=0, the pointer contains the address of the destination for the data to be stored.

Table continues on the next page...

Table 10-32. STORE/SEQ STORE command field descriptions (continued)

Field	Description
	If SGF=1, the pointer points to a Scatter/Gather Table, which defines the destinations for the data to be stored. Note that SGF should not be set to 1 when using SRC values 41h or 42h. Doing so will cause an error to be generated. **NOTE:** If the IMM bit is set, it is an error for the SGF bit to be set. If CTYPE = 01011b (SEQ STORE), this bit is the Variable Length Flag (VLF). If VLF=0, the LENGTH field indicates the length of the data. If VLF=1, the data length is variable. CAAM uses the length in the Variable Sequence Out Length register rather than the value the LENGTH field. However, an error will be generated if the values in the VSOL register and OFFSET field are not a valid combination as indicated in the table Table 10-33.
23 IMM	Immediate data. If IMM=0 : Data to be stored is found at the location specified by the SRC field. If IMM=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, as necessary, to the nearest 4-byte word. For SEQ STORE, the data immediately follows the command; for STORE, the data immediately follows the pointer. **NOTE:** It is an error if the IMM bit is set when the SGF bit is set. However, the destination of a SEQ STORE can be 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 (e.g. CONTEXT) of the data to be stored. See Table 10-33 for a list of supported sources. If IMM = 1 the data to be stored is located as immediate data within the command. Although the SRC field does not specify the source of the data, the SRC field still determines whether the immediate data is treated as message data or control data. When CAAM is configured for big endian operation, message data and control data are treated the same. When CAAM is configured for little endian operation, control data is byte swapped within words as the immediate data is stored into memory but message data is stored as-is, without byte swapping. SRC = 00h will cause the immediate data to be treated as control data, so when CAAM is configured for little-endian operation the data will be byte-swapped within words before it is written to memory. SRC = 7Eh will cause the immediate data to be treated as message data, so the data will be written as-is, without byte swapping. The use of any other SRC value with IMM=1 will cause an error to be generated.
15-8 OFFSET	OFFSET defines the start point for reading within the SRC. For example, if the SRC indicates a context register, the offset can be used to indicate that the data should be read from the fourth byte 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 Table 10-33 for the legal combinations of OFFSET and LENGTH values.
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 Table 10-33 for the legal combinations of OFFSET and LENGTH values.
Additional words of STORE command:	
31-0 POINTER	This field is a pointer to the address in memory where the data is to be stored. **NOTE:** 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 which use the pointers previously specified for the job and shared descriptors. (Type 40h requires a pointer for the STORE command.)
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 10-33. 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 8/0 bytes 4/4 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 8/0 bytes 4/4 bytes	KEYS1	Class 1 Key Size Register	
	10			KEYS2	Class 2 Key Size Register	
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DDAR	DECO Descriptor Address Register	
02	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	DATAS1	Class 1 Data Size Register	
	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 Table 10-88, TEST CONDITION field, TEST CONDITION bits when JSL = 0) as well as CAAM's current command index. The status is in the left four bytes of this register. The right four bytes contain the number of bytes written to the SEQ OUT PTR address.
03	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	C1ICVS	Class 1 ICV Size Register	
	10			C2ICVS	Class 2 ICV Size Register	
04	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DDID	DECO DID Register	Consists of the following fields: Shared DID, Job DID, TZ/SDID, Trusted DID.
06	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CCTRL	CHA Control Register	

Table continues on the next page...

Table 10-33. 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
07	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	ICTRL	IRQ Control Register	
08	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CLRW	Clear Written Register	
08	11	Control	0-32 0-7 bytes	MATH0W	DECO Math Register 0 (Words)	1, 2
09	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CSTA	CCB Status and Error Register	
	11	Control	0-24 0-7 bytes	MATH1W	DECO Math Register 1 (Words)	1, 2
0A	11	Control	0-16 0-7 bytes	MATH2W	DECO Math Register 2 (Words)	1, 2
0B	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	AADSZR	AAD Size Register	1, 2
	11	Control	0-8 0-7 bytes	MATH3W	DECO Math Register 3 (Words)	1, 2
10	11	Control	16/0 bytes 32/0 bytes 48/0 bytes 64/0 bytes	GTR	Gather Table Registers	
10	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKASZ	PKHA A Size Register	-
11	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKBSZ	PKHA B Size Register	-
12	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKNSZ	PKHA N Size Register	-

Table continues on the next page...

Table 10-33. 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
13	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKESZ	PKHA E Size Register	-
20	01	Message	0-128/ 0-128 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.
20	11	Control	16/0 bytes 32/0 bytes 48/0 bytes 64/0 bytes	STR	Scatter Table Registers	
30	11	Control	0-32 0 bytes	MATH0DW	DECO Math Register 0 (Double Word)	1, 3, 2
31	11	Control	0-24 0 bytes	MATH1DW	DECO Math Register 1 (Double Word)	
32	11	Control	0-16 0 bytes	MATH2DW	DECO Math Register 2 (Double Word)	
33	11	Control	0-8	MATH3DW	DECO Math Register 3 (Double Word)	
38	11	Control	0-32 0-7 bytes	MATH0B	DECO Math Register 0 (Bytes)	1, 4
39	11	Control	0-24 0-7 bytes	MATH1B	DECO Math Register 1 (Bytes)	
3A	11	Control	0-16 0-7 bytes	MATH2B	DECO Math Register 2 (Bytes)	
3B	11	Control	0-8 0-7 bytes	MATH3B	DECO Math Register 3 (Bytes)	
40	01	Message	0-128/0-128 bytes	KEY1	Class 1 Key Register	If the corresponding Key Size register has not been written, the STORE or SEQ STORE command may be used to store the key register into memory. After
	10	Message	0-128/0-128 bytes	KEY2	Class 2 Key Register	

Table continues on the next page...

Table 10-33. 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
						the key size has been written, the key register can be stored to memory only via a FIFO STORE or SEQ FIFO STORE command.
	11	Control	0-64/ 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 values in the LENGTH and OFFSET field are specified in 4-byte words. offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 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 descriptor buffer.			
41	11	Control	0-64/ 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 (including 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, an error will be generated for a STORE command with SRC=41h. Note that SGF should not be set to 1. Doing so will cause an error to be generated. The values in the LENGTH and OFFSET field are specified in 4-byte words. offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 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 Figure 10-8.			
42	11	Control	0-64/ 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 shared descriptors. This overwrites the shared descriptor in memory, using the address from which the shared descriptor was fetched. Note that a STORE with SRC=42h results in an error if there is no shared descriptor. Even if there is a shared descriptor in the original descriptor, an error is generated if there has been a non-local jump to			

Table continues on the next page...

Table 10-33. 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
			another descriptor or an in-line descriptor is being executed, and that descriptor attempts a STORE with SRC=42h. Note that SGF should not be set to 1 for SRC values 41h or 42h. Doing so will cause an error to be generated. Since no pointer is used, this is a one-word command. The values in the LENGTH and OFFSET field are specified in 4-byte words. To correctly use sharing flows (wait or serial) in CAAM, if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, CAAM will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete. offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 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 Figure 10-8.			
All combinations of SRC and CLASS that do not appear in Table 10-33 are reserved.						

1. Because the math registers are in contiguous addresses, it is possible to store more than one math register simultaneously.
2. When this source is used, the data stored from the Math register will be treated as words.
3. When this source is used, the data stored from the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers.
4. When this source is used, the data stored from the Math register will be treated as bytes.

10.6.14 FIFO STORE command

NOTE

In the following discussion, FIFO STORE command refers to both the SEQ and non-SEQ forms of the command.

FIFO STORE commands are used 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 SGT bit is replaced by the VLF bit. See SEQ vs non-SEQ commands.

Note that data output by means of the output data FIFO is considered message data. Therefore, it is byte-swapped, half-word-swapped, full-word-swapped, 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 may be encrypted as a Black Key, so all crypto operations should be completed prior to this command being executed.
- PKHA registers, other than the E-Memory.
- PKHA E Memory. This data is encrypted as a Black Key prior to being written to memory.
- 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.
- Data in the input sequence or in the input data FIFO

Note that even though this command is not a store check point, it does not start if a prior STORE or FIFO STORE of any type has yet to be scheduled. This command is a done checkpoint if asked to encrypt a key, because it has to wait until both class CHAs are done. The FIFO STORE command will block if the internal CCB DMA is not available when storing Class 1 or Class 2 keys.

The FIFO LOAD command supports 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 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 command and the REW field in the SEQ OUT PTR command
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. Note that scatter tables may be read while skipping if the sequence was defined with the SGF bit set in the SEQ OUT PTR command. However, no data will be written while skipping.

Table 10-34. 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

15	14–10	9–0
PKLE	reserved	LENGTH *when FIFO STORE data type selects a PKHA register*
LENGTH *(16 bits, when PKHA is not selected)*		

Table continues on the next page...

Table 10-34. FIFO STORE command format (continued)

Additional words of FIFO STORE command:
Pointer (one word, see Address pointers)
EXT LENGTH (Present if EXT = 1) (one word)

Table 10-35. FIFO STORE command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=01100b : FIFO STORE command If CTYPE=01101b : SEQ FIFO STORE command
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 Table 10-36.
24 SGF or VLF	Scatter/Gather table Flag (SGF) or Variable Length Flag (VLF). Meaning depends on CTYPE. If CTYPE = 01100 (FIFO STORE), this bit is the Scatter/Gather table Flag (SGF). If SGT=0 : The pointer points to actual data. If SGT=1 : The pointer points to a scatter/gather table. If CTYPE = 01101 (SEQ FIFO STORE), this bit is the Variable Length Flag (VLF). If VLF=0 : The number of bytes of data to be stored is specified in the LENGTH (if EXT=0) or EXT LENGTH (if EXT=1) field. If VLF=1 : The data length is variable. The number of bytes of data to be stored is specified in the VSOL register. The LENGTH field is ignored. **NOTE:** It is legal to set VLF=1 when storing a key **NOTE:** It is illegal to set VLF=1 when EXT=1.
23 CONT	Continue If CONT=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. If the read from the output FIFO ends with the last byte of the dword, that dword is always popped. If CONT=1 : 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. **NOTE:** If this bit is set when there is no remaining data, subsequent operations may not work as expected.
22 EXT	Use Extended Length If EXT=0 : Output data length is solely determined by the 16-bit LENGTH field in the first word of the command. If EXT=1 : Output data length is determined by the 32-bit EXT LENGTH field. **NOTE:** It is illegal to set VLF=1 when EXT=1.
21-16 Output Data Type	This field identifies the type of data that the output data FIFO stores. See Table 10-36 for a list of the supported types.
15 PKLE	PKHA Little Endian data store If PKLE=0: data is not swapped as stored from a PKHA Memory If PKLE=1: data is swapped as stored from a PKHA Memory.

Table continues on the next page...

Table 10-35. FIFO STORE command field descriptions (continued)

Field	Description
(If data type selects a PKHA register)	This feature is intended to support curves like Curve25519 and Ed25519, which are typically represented as little-endian byte strings, but not for Weierstrass curves, which are typically represented as big-endian byte strings. PKLE must be set to 0 for black (encrypted) keysand when storing from PKHA E RAAM. **NOTE:** In this instance of CAAM DECO will report an 'Invalid FIFO STORE command' error if PKLE is set and the length is encoded in the first word. Instead set the EXT bit and encode length in the extension word, or set the VLF bit and utilize VSOL.
15-0 *(for PKHA data type: 9-0)* LENGTH	The length of the data to be stored. If EXT=0 : The LENGTH field specifies the number of bytes to store. If EXT=1 : The EXT FIELD specifies the number of bytes to store. The LENGTH field is ignored.
Additional words of the FIFO STORE command:	
POINTER	Address pointer where to store the data in memory. **NOTE:** 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.
EXT LENGTH	Extended length field. If EXT=0 : This field is not present. The LENGTH field specifies the number of bytes of data to be stored. If EXT=1 : The EXT LENGTH field specifies the number of bytes of data to be stored. The LENGTH field in the first word of the command is ignored.

Table 10-36 lists the various built-in FIFO STORE output data types.

Table 10-36. FIFO STORE output data type field

Bits 21-16 (hex)	Meaning	Comment
00	PKHA A0	**NOTE:** The appropriate size register is automatically written. A FIFO STORE from a PKHA register should never be attempted with size greater than the PKHA register size.
01	PKHA A1	
02	PKHA A2	
03	PKHA A3	
04	PKHA B0	
05	PKHA B1	
06	PKHA B2	
07	PKHA B3	
08	PKHA N	
0C	PKHA A	
0D	PKHA B	

Table continues on the next page...

Table 10-36. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
10	AFHA S-Box, encrypted using AES-CCM with the job descriptor key encryption key.	The S-Box will be stored in plaintext form (rather than encrypted) if the S-Box was loaded with a KEY command with PTS=1 or if a key was loaded into the Class 1 Key register and AFHA was run in INIT mode to create an S-Box.
11	AFHA S-Box, encrypted using AES-CCM with the trusted descriptor key encryption key.	Available only to trusted descriptors. The S-Box will be stored in plaintext form (rather than encrypted) if the S-Box was loaded with a KEY command with PTS=1 or if a key was loaded into the Class 1 Key register and AFHA was run in INIT mode to create an S-Box.
12	PKHA E, encrypted using AES-CCM with the job descriptor key encryption key	
13	PKHA E, encrypted using AES-CCM with the trusted descriptor key encryption key.	Available only to trusted descriptors.
14	Key Register, encrypted using AES-CCM with the job descriptor key encryption key.	The AUX field determines the source register for the FIFO STORE. • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.
15	Key register, encrypted using AES-CCM with the trusted descriptor key encryption key.	Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE. • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.
16	Class 2 Key Register Derived HMAC Key, encrypted using AES-CCM with the job descriptor key encryption key.	For performance and security, use of Derived HMAC Key is highly recommended. Details about such keys can be found in Using the MDHA Key Register with Derived HMAC Keys. The length of Derived HMAC Key is twice the length of the chosen MDHA algorithm's running digest (see MDHA use of the Context Register). If the Class 2 Key register was loaded with a Derived HMAC Key using a KEY command with PTS=1, or if a key was loaded into the Class 2 Key register and then the MDHA was run in INIT mode to create a Derived HMAC Key, the Class 2 Key register will be stored in plaintext form.
17	Class 2 Key Register Derived HMAC Key, encrypted using AES-CCM with the trusted descriptor key encryption key.	Available only to trusted descriptors. The comments for type 16h apply here as well.
20	AFHA S-Box, encrypted using AES-ECB with the job descriptor key encryption key	The S-Box will be stored in plaintext form (rather than encrypted) if the S-Box was loaded with a KEY command with PTS=1 or if a key was loaded into the Class 1 Key register and AFHA was run in INIT mode to create an S-Box.

Table continues on the next page...

Table 10-36. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
21	AFHA S-Box, encrypted using AES-ECB with the trusted descriptor key encryption key.	Available only to trusted descriptors. The S-Box will be stored in plaintext form (rather than encrypted) if the S-Box was loaded with a KEY command with PTS=1 or if a key was loaded into the Class 1 Key register and AFHA was run in INIT mode to create an S-Box.
22	PKHA E, encrypted using AES-ECB with the job descriptor key encryption key	-
23	PKHA E, encrypted using AES-ECB with the trusted descriptor key encryption key.	Available only to trusted descriptors.
24	Key Register, encrypted using AES-ECB with the job descriptor key encryption key.	The AUX field determines the source register for the FIFO STORE. • AUX = 01 selects the Class 1 key register to be stored. • AUX = 10 selects the Class 2 key register to be stored. AUX values 00 and 11 are illegal.
25	Key Register, encrypted using AES-ECB with the trusted descriptor key encryption key.	Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE. • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.
26	Class 2 Key Register Derived HMAC Key, encrypted using AES-ECB with the job descriptor key encryption key.	For performance and security, use of a Derived HMAC Key is highly recommended. Details about such keys can be found in Using the MDHA Key Register with Derived HMAC Keys. The length of the derived HMAC key is twice the length of the chosen MDHA algorithm's running digest (see MDHA use of the Context Register). If the Class 2 Key register was loaded with a Derived HMAC Key using a KEY command with PTS=1, or if a key was loaded into the Class 2 Key register and then the MDHA was run in INIT mode to create a Derived HMAC Key, the Class 2 Key register will be stored in plaintext form.
27	Class 2 Key Register Derived HMAC Key, encrypted using AES-ECB with the trusted descriptor key encryption key.	Available only to trusted descriptors. The comments for type 26h apply here as well.
30	Message Data	
34	Store the specified amount of data to be obtained from RNG to memory.	**NOTE:** The Class 1 Data Size Register is automatically written and extended lengths are illegal. The different types of random data that can be generated are: • Random data with no restriction • Nonzero Random data • Odd Parity Random data.

Table continues on the next page...

Table 10-36. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
		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.	In addition to the comments for type 34h, there is no pointer and it is illegal to use type 35h with SEQ FIFO STORE.
3E	Meta Data	For this output data type, CONT and EXT must both be 0. Either bit set to a 1 generates an error. This type can be used only with SEQ FIFO STORE; an error is generated if this output data type is used with FIFO STORE. Length can be specified in the command (VLF = 0) or in the Variable Sequence Out Length register (VLF = 1). If VLF = 1, the length must fit in the lower 16 bits of the VSIL register or an error is generated. The AUX bits control the behavior of the SEQ FIFO STORE command as follows: 00 Use the DECO alignment block to move the specified number of bytes from the input FIFO to the output FIFO and store them to the output frame. This variant of the command is used when handling meta data that has already been read. An example of this would be for a shared descriptor where the RIF bit is set. 01 The same as 00, except that the VSIL is decremented by the specified length. This form should be used when the RIF bit is set and the VSIL contains the input frame length of the packet. If the VSIL were not decremented in this case, the descriptor would have to subtract the meta data length from the VSIL register. 10 Load the specified number of bytes from the input frame, as defined by a prior SEQ IN PTR command, to the input FIFO and decrement the Sequence In Length by this number of bytes. Move these bytes to the output FIFO by means of the DECO alignment block and store them to the output frame. This variant of the command is used when handling meta data that precedes the packet data. 11 The same as 10, except that the Sequence In Length is not decremented. This form should be used when moving meta data that follows the packet data. Normally the length of trailing meta data has to be subtracted from the input frame length prior to running a protocol so that the protocol knows how long the packet is. When using the AUX = 11 variant the descriptor does not have to add the meta data length back to the Sequence In Length before executing the SEQ FIFO STORE Meta Data command.
3F	Skip	Skip over the specified length in memory without using bus cycles. Permitted to be used only by SEQ FIFO STORE.
NOTE: AUX must be set to 00 except when otherwise specified above. All combinations of output data type and AUX values not specified are reserved.		

10.6.15 MOVE, MOVEB, MOVEDW, and MOVE_LEN commands

NOTE

In this section "Move Command" is used to refer to the MOVE, MOVEB, MOVEDW, and MOVE_LEN forms of the command.

The MOVE command is used to copy data between two resources internal to a DECO/CCB. This allows data to be put in the proper registers without having to store data to external memory and then load it.

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 Table 10-40). The MOVE command has a limited number of sources and destinations as indicated in the SRC and DST field descriptions below.

NOTE

MOVE cautions and restrictions:

- Keys can't be copied from a key register by means of a MOVE command if the corresponding key size register has been written.
- Observe the cautions noted in the "RJD" field of SEQ IN PTR command if using a MOVE command in a Replacement Job Descriptor.
- Moves 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.

When moving data to or from the Descriptor Buffer or a MATH register, MOVEB treats data as 32-bit words in those cases that MOVE treats data as bytes, and MOVEB treats data as bytes when MOVE treats data as words if byte swapping is enabled, but is identical to the MOVE command if byte swapping is not enabled. The MOVEDW command and the MOVE_LEN command for dwords always treat data as double words (i.e. 64 bits) and, by default, perform word swapping. MOVEDW and MOVE_LEN for dwords can be configured to not do word swapping. The MOVEDW command and the MOVE_LEN command for dwords never do byte swapping.

NOTE: For MOVEDW or MOVE_LEN for dwords, with one exception, the offset must be a multiple of dwords (8 bytes). The one exception is when the offset is into the descriptor buffer, in which case the offset is allowed to be a multiple of words (4 bytes). If the source is the Output FIFO dword moves will always result in an error if the OFIFO offset is not zero.

In the MOVE, MOVEB, or MOVEDW command, the LENGTH field specifies the amount of data to be moved. The MOVE_LEN command is identical to the MOVE, MOVEB, and MOVEDW commands except that the length of the data being moved is specified in a MATH register, rather than specified as a constant in the LENGTH field. In the MOVE_LEN command, the MRSEL (Math Register Select) field, the TYPE field, and a reserved field replace the MOVE command's LENGTH field.

The AUX field is used to select among a number of different options, depending on the values in the SRC and DST fields (see the table Table 10-40 below).

The MOVE command will block if the CCB DMA is busy. Other conditions where the MOVE command will block include:

- The SRC is context and the corresponding class CHA is not done or there is a data in flight to either context register.
- The SRC is the Output FIFO but a request for the external DMA to pull data from the output FIFO is pending.
- The DST is a context register and there is data in flight to either context register.
- The DST is the input data FIFO but there is data in flight to the input data FIFO.
- The DST is the input data FIFO for either of the C1 or C2 alignment blocks and an NFIFO entry is to be written and there is a context load pending.

NOTE

For this device, the default is that byte swapping is enabled. This means that MOVE, MOVEB, and MOVE_LEN (when the TYPE is 00 or 10) will swap bytes.

NOTE

For this device, the default is that word swapping is enabled. This means that MOVEDW and MOVE_LEN (when the TYPE is 01) will swap words.

NOTE

This device does not prevent the byte and word swapping defaults from being overridden. However, care should be used when changing this behavior.

The Output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the Output FIFO so each can track separately allowing consumption of data at different rates. However, the only time these indices track separately is when the NFIFO is consuming data from the Output FIFO. Therefore, when using a move command to extract data from the Output FIFO, it is critical that the descriptor writer know whether the indices are

tracking together and, if not, which index needs to be used to obtain the desired data. Note that this is an extremely unusual circumstance which most descriptor writers will seldom, if ever, encounter.

In prior versions of CAAM, different entities handled the move from the Output FIFO depending on alignment. If the OFFSET specified was a multiple of words and the current OFIFO offset was 0, then the CCB DMA handled the move. Otherwise, the external DMA handled the move. The DMA used determined the index that was used to access the data in the Output FIFO. This led to significant confusion about which data was being read. In this version of CAAM, the CCB DMA handles all moves from the Output FIFO, eliminating the confusion. However, it is possible for the descriptor to manipulate the index.

NOTE

It is possible to make the earlier behavior forward compatible by careful descriptor construction.

- To ensure that the move reads from the index where the external DMA left off, perform a LOAD IMM to the DECO Control Register to reset the CHA pointer in the Output FIFO. This has the effect of setting the shared index to the same value as the index used by the external DMA. While this will lose the current index of the CHA pointer, the move will get the expected data. (Remember that this is only necessary when the two indices are different. If the amount snooped and the amount read by the external DMA are the same, the indices will be the same.)
- To ensure that the move reads from the index where the NFIFO left off, ensure that the OFIFO offset is 0 and that the OFFSET in the move command is 0.

The OFIFO offset is used in two ways. First, it is used by DECO to tell the external DMA where in the 8-byte interface to the Output FIFO to start reading. Second, it is used by the CCB DMA to know where in the 8-byte interface to the Output FIFO to start reading. However, the two DMAs use different indices to access the Output FIFO so that the offset could be referencing different dwords. While those indices are usually synchronized, they can become different when the NFIFO has pulled data from the Output FIFO. It is therefore critical that the descriptor writer keep track of where each index is when moves from the Output FIFO follow snooping.

Table 10-37. MOVE, MOVEB, MOVEDW, and MOVE_LEN command format

	31–27	26–25	24	23–20	19–16
	CTYPE = 01111, 00110, 00111 or 01110	AUX	WC	SRC	DST
Fields as they appear in the MOVE,	**15–8**			**7-0**	
	OFFSET			LENGTH	

Table continues on the next page...

Table 10-37. MOVE, MOVEB, MOVEDW, and MOVE_LEN command format (continued)

MOVEB, or MOVEDW command:				
Fields as they appear in the MOVE_LEN command:	**15–8**	**7-6**	**5-2**	**1-0**
	OFFSET	TYPE	Reserved	MRSEL

Table 10-38. MOVE command field descriptions

Field	Description
31-27 CTYPE	Command Type 01111 - MOVE. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the LENGTH field. If byte swapping is enabled, MOVE swaps bytes within words in certain cases (see table Table 10-39). 00111 - MOVEB. When byte swapping is not enabled, the legal MOVEB moves are identical to the corresponding MOVE moves. However, when byte swapping is enabled, the MOVEB moves byte swap within words when the corresponding MOVE moves do not swap and vice versa (see table Table 10-39). 00110 - MOVEDW. Move Double Words. Performs an internal move between two internal DECO/CCB locations. If word swapping is not enabled, the legal MOVEDW moves are identical to the corresponding MOVE moves when byte swapping is disabled. If word swapping is enabled for the descriptor, the MOVEDW command swaps the order of the two words in a double word. No byte swapping is done. 01110 - MOVE_LEN. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the MATH register selected by the MRSEL field. Byte or word swapping may be done based on the endianness settings and the value in the TYPE field.
26-25 AUX	AUX bits are used for some SRC and DST combinations to specify additional options. See table Table 10-40 below.
24 WC	Wait for Completion 0 - Do not Wait for Completion 1 - Wait for Completion. Causes the MOVE command to stall until the move operation completes. This is necessary when the data to be moved must be in place before a subsequent command executes. While it is sometimes possible to know a priori that the MOVE command will complete prior to reaching the subsequent command in question, such completion can not always be guaranteed.
23-20 SRC	Source. This specifies the internal source of data that will be moved. See table Table 10-42 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 10-40 and Table 10-41 indicate which source and destination combinations are permitted.
19-16 DST	Destination. This specifies the internal destination of the data that will be moved. See table Table 10-43 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 10-40 and Table 10-41 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 in table Table 10-41. 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. The offset is part of the length when going from one of the alignment blocks to the output FIFO. This allows a 16-bit length to be specified. For MOVEDW and MOVE_LEN for dwords, the OFFSET must be a multiple of 8 bytes unless the OFFSET is into the Descriptor Buffer, in which case the OFFSET must be a multiple of 4 bytes.

Table continues on the next page...

Table 10-38. MOVE command field descriptions (continued)

Field	Description
7-0 LENGTH	Length for internal move. (in bytes, 128 max) This field appears only in the MOVE, MOVEB, or MOVEDW forms of the command. In the MOVE_LEN form of the command this field is replaced by reserved bits and the MRSEL field and TYPE field, as shown below.
Note that in the MOVE_LEN form of the command the LENGTH field is replaced by the following three fields:	
7-6 TYPE	Type of the data items that are to be moved. 00 - Data is treated the same as in the MOVE command 01 - Data is treated as dwords the same way as in the MOVEDW command 10 - Data is treated as bytes the same way as in the MOVEB command 11 - Reserved and reported as an error
5-2	These bits are reserved in the MOVE_LEN form of the command. These bits, the TYPE field 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, TYPE field, and the reserved bits above replace the LENGTH field that appears in the MOVE, MOVEB, and MOVEDW forms 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 or any of the alignment blocks 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 10-39. Byte swapping in move commands

When byte swapping is enabled, this table indicates when bytes within a word are swapped.

Legend:

- M refers to the MOVE command and the MOVE_LEN command when TYPE=00
- B refers to the MOVEB command and the MOVE_LEN command when TYPE=10
- Swap: indicates the move will swap bytes within words
- Not: indicates the move will not swap bytes within words
- Err: indicates the move command will generate an error

DST --> SRC | v	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: Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: PKHA A RAM (automatically flushed)	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
0h: C1 Context 1h: C2 Context	M:Not B:Err		M:Swap B:Not	M:Not B:Swap	M:Not B:Err					
2h: Output FIFO	M:Not B:Err	M:Err B:Err								

Table continues on the next page...

Table 10-39. Byte swapping in move commands (continued)

3h: Descr Buffer	**M:Swap** **B:Not**	**M:Err** **B:Err**		**M:Swap** **B:Not**	**M:Not** **B:Swap**	**M:Swap** **B:Not**
4h: Math Reg 0 **5h: Math Reg 1** **6h: Math Reg 2** **7h: Math Reg 3**	**M:Not** **B:Swap**					
8h: DECO Alignment Block (flushed)	**M:Not** **B:Err**	**M:Swap** **B:Not**	**M:Not** **B:Swap**	**M:Err** **B:Err**	**M:Not** **B:Err**	**M:Err** **B:Err**
9h: Class 1 or Class 2 Alignment Block						
Ah: DECO, Class 1 or Class 2 Alignment Block						
Dh: Class 1 Key **Eh: Class 2 Key**				**M:Not** **B:Err**		

Table 10-40. Usage of the AUX field in move commands

DST --> **SRC** **|** **v**	**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: Data FIFO**	**9h: Class 2 Input Data FIFO**	**Ah: Input Data FIFO (no NFIFO entries)**	**Ch: PKHA A RAM (automatically flushed)**	**Dh: C1 Key** **Eh: C2 Key**	**Fh: Aux Data FIFO**
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 selects offset into Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes	AUX_{MS} : Flush AUX_{LS} : Last	AUX_{LS}: Last			AUX_{LS}=0 : OFFSET field into Context Reg AUX_{LS}=1 : OFFSET into Key Reg	
2h: Output FIFO		move not allowed								
3h: Descr Buffer	AUX selects offset into Context Register 00: 0 bytes		move not allowed	AUX selects offset into Math Register 00: 0 bytes						

Table continues on the next page...

Table 10-40. Usage of the AUX field in move commands (continued)

	01: 16 bytes 10: 32 bytes 11: 48 bytes			01: 4 bytes 10: 6 bytes 11: 7 bytes						
4h: Math Reg 0 **5h: Math Reg 1** **6h: Math Reg 2** **7h: Math Reg 3**	AUX selects offset into Math Register 00: 0 bytes> 01: 4 bytes 10: 6 bytes 11: 7 bytes		AUX selects offset into Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes	AUX selects offset into the source Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes						
8h: DECO Alignment Block (automatically flushed)					move not allowed					move not allowed
9h: Class 1 or Class 2 Alignment Block	AUX_{MS} : Flush AUX_{LS} =0 : Source is Class 2 Alignment Block AUX_{LS} =1 : Source is Class 1 Alignment Block								AUX_{MS} : Flush AUX_{LS} =0 : Source is C2 Align Block AUX_{LS} =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 Alignment Block 00: DECO Alignment Block 01: C1 AB 10: C2 AB 11: error	
Dh: Class 1 Key **Eh: Class 2 Key**	Determines SRC/DST offset				AUX_{MS} : Flush AUX_{LS} : Last	AUX_{LS}: Last			Determines SRC/DST offset	

Table 10-41. Usage of the OFFSET field in move commands

DST --> SRC | v	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: PKHA A RAM (always flushed)	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	OFFSET field is used for Descriptor Buffer (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC					AUX_{LS}=0: OFFSET field is used for Context Reg AUX_{LS}=1: OFFSET field is used for Key Reg	OFFSET field is used for SRC
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	Error generated if OFFSET≠0
3h: Descr Buffer	OFFSET field is used for SRC (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC	move not allowed	OFFSET field is used for SRC					OFFSET field is used for SRC	OFFSET field is used for SRC
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	OFFSET field is used for DST	OFFSET field is used for SRC					OFFSET field is used for DST	OFFSET field is used for SRC
8h: DECO Align Block (flushed)	OFFSET field is used for DST	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	move not allowed					OFFSET field is used for DST	move not allowed
9h: Class 1 or Class 2 Alignment Block										
Ah: DECO, Class 1 or Class 2 Alignment Block										

Table continues on the next page...

Table 10-41. Usage of the OFFSET field in move commands (continued)

Dh: Class 1 Key **Eh: Class 2 Key**	AUX_{LS} =0 : SRC; else DST	SRC	DST	SRC	SRC	AUX_{LS} =0 : SRC; else DST	SRC

Table 10-42. Move sources

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	A MOVE command that reads past the end of MATH Register 3 will return 0s for all data past Math Register 3.
5h	Math Register 1	
6h	Math Register 2	
7h	Math Register 3	
8h	DECO Alignment Block (always Flushed)	Input to the DECO Alignment Block is specified by an NFIFO entry which is automatically generated if Automatic NFIFO entries are enabled.
9h	Class 1 or Class 2 Alignment Block	The selection of the Class 1 or Class 2 Alignment Block is determined by the least-significant bit of the AUX field: • A_{LS} = 0 selects C2 Alignment Block • A_{LS} = 1 selects C1 Alignment Block Input to the Class 1 or Class 2 Alignment Block is specified by an NFIFO entry, which is automatically generated if Automatic NFIFO entries are enabled. The most-significant bit of the AUX field must be set (causing a FLUSH) only if the destination is the Output Data FIFO.
Ah	DECO, Class 1 or Class 2 Alignment Block, as specified via the AUX field.	no NFIFO entry generated; AUX = 00b: use DECO Alignment Block AUX = 01b: use Class 1 Alignment Block AUX = 10b: use Class 2 Alignment Block AUX = 11b: error
Dh	Class 1 Key	Error if C1 Key Size has been written either directly or by the KEY command and not cleared.
Eh	Class 2 Key	Error if C2 Key Size has been written either directly or by the KEY command and not cleared.
All other values are reserved		

Table 10-43. Move destinations

Value	Move Destination	Notes
0h	Class 1 Context	—
1h	Class 2 Context	—

Table continues on the next page...

Table 10-43. Move destinations (continued)

Value	Move Destination	Notes
2h	Output Data FIFO	
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 NFIFO entries are enabled, the entries are generated for a Class 1 CHA.
9h	Input Data FIFO (C2)	If Automatic NFIFO entries are enabled, the entries are generated for a Class 2 CHA.
Ah	Input Data FIFO	No NFIFO entry generated
Ch	PKHA A	If Automatic NFIFO entries are enabled, the entries are generated for the PKHA A RAM and the Flush bit is automatically set.
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 should 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		

10.6.16 ALGORITHM OPERATION command

The OPERATION command (CTYPE = 10000) defines the type of CHA operation CAAM performs. Setting OPTYPE = 010b or 100b specifies a type of OPERATION command called an ALGORITHM OPERATION. Setting OPTYPE = 000, 110, or 111 specifies a different type of OPERATION command called a PROTOCOL OPERATION.Setting OPTYPE = 001 specifies a third type of OPERATION command called a PKHA OPERATION. More than one OPERATION command can be used in a descriptor, allowing Class 1 and Class 2 operations to be specified separately, i.e., operations can range from performing a single command using a single CHA to performing complex operations involving multiple CHAs and/or multiple commands.

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 10-44. ALGORITHM OPERATION command format

	31-27	26-24	23-16

Table continues on the next page...

Table 10-44. ALGORITHM OPERATION command format (continued)

	CTYPE = 10000b			OPTYPE = 010b or 100b			ALG				
	15-13	12-4							3-2	1	0
format as it appears for CHAs other than RNG:	Reserved	AAI							AS	ICV	ENC
	15-13	12	11	10	9	8	7-6	5-4	3-2	1	0
format as it appears for RNG:	Reserved	SK	AI	PS	OBP	NZB	Reserved	SH	AS	PR	TST

Table 10-45. ALGORITHM OPERATION command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in the field descriptions below. If OPTYPE = 010b : Class 1 algorithm operation If OPTYPE = 100b : Class 2 algorithm operation If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command. If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION Command.

23-16
ALG
	Algorithm
	This field specifies the algorithm that is to be used for the operations.
	• If OPTYPE = 010b (Class 1 algorithm)
	  • If ALG=10h : AES
	  • If ALG=20h : DES
	  • If ALG=21h : 3DES
	  • If ALG=30h : ARC4
	  • If ALG=50h : RNG
	  • All other values are reserved.
	• If OPTYPE = 100b (Class 2 algorithm)
	  • If ALG=40h : MD5
	  • If ALG=41h : SHA-1
	  • If ALG=42h : SHA-224
	  • If ALG=43h : SHA-256
	  • All other values are reserved.

15-13	Reserved
12-4 AAI	Additional Algorithm Information This field contains additional mode information that is associated with the algorithm that is being executed. See the tables below for details specific to individual algorithms. 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.

Table continues on the next page...

Table 10-45. ALGORITHM OPERATION command field descriptions (continued)

Field	Description
	For RNG OPERATION commands the AAI field is interpreted as shown in the shaded SK, AI, PS, OBP, NZ and SH fields below.
3-2 AS	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. 00 Update 01 Initialize 10 Finalize 11 Initialize/finalize
1 ICV	ICV Checking 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.
0 ENC	Encrypt/Decrypt 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. 0 Decrypt 1 Encrypt
The rows below describe how bits 12-0 are interpreted for RNG commands.	
12 SK (RNG only)	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 (RNG only)	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 (RNG only)	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 (RNG only)	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 (RNG only)	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.

Table continues on the next page...

Table 10-45. ALGORITHM OPERATION command field descriptions (continued)

Field	Description
(RNG only)	
5-4 SH (RNG only)	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 All other codes reserved
3-2 AS (RNG only)	Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown in Table 10-49.
1 PR (RNG only)	Prediction Resistance. For RNG OPERATION commands this bit is interpreted as shown in Table 10-50.
0 TST (RNG only)	Test Mode Request. For RNG OPERATION commands this bit is interpreted as shown in Table 10-51.

Table 10-46. AAI Interpretation for AES modes

AAI Interpretation for AES Modes (See AES accelerator (AESA) functionality)				
Code	**Interpretation**		**Code**	**Interpretation**
00h	CTR (mod 2^{128})		80h	CCM (mod 2^{128}), [1]
10h	CBC		90h	GCM (mod 2^{32})[1]
20h	ECB			
30h	CFB128			
40h	OFB			
60h	CMAC			
70h	XCBC-MAC		12h	CBC-CS2 (CTS)
The codes listed above are mutually exclusive, which means that they cannot be ORed with each other. All unlisted code points (except those created by OR-ing AAI with the Decrypt Key selector bit described below) are reserved in this version of CAAM.				
Note that for AES the MSB of AAI is the DK (Decrypt Key) bit. Setting the DK bit (that is, OR-ing 100h with any AES code above) causes the Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in AES accelerator (AESA) functionality. When a Class 2 OPERATION command specifies AES with CMAC or XCBC-MAC, it may be accompanied by a Class 1 OPERATION command specifying AES, if (and only if) the Class 1 OPERATION command specifies a Confidentiality-only mode. Specifying a Class 2 AES OPERATION command in concert with a Class 1 AES Operation command specifying either CCM or GCM is not permitted and will result in an error. Combo modes CBC-XCBC-MAC, CTR-XCBC-MAC, CBC-CMAC,				

Table 10-46. AAI Interpretation for AES modes

AAI Interpretation for AES Modes (See AES accelerator (AESA) functionality)
CTR-CMAC-PDCP, and CTR-CMAC were specified for the built-in IPsec and PDCP protocol but are available for general use. With the extension to AES permitting simultaneous Class 1 Confidentiality-only and Class 2 Integrity OPERATION, these Combo modes are no longer recommended and may be deprecated in the future.

1. CCM and GCM use the key in the Class 1 Key Register.

Table 10-47. AAI Interpretation for DES modes

AAI Interpretation for DES modes (See Data encryption standard accelerator (DES) functionality)				
Code	**Interpretation**		**Code**	**Interpretation**
10h	CBC		30h	CFB8
20h	ECB		40h	OFB
The codes listed above are mutually exclusive, which means that they cannot be ORed with each other.				
80h ORed with any DES code above: Check odd parity				

Table 10-48. AAI Interpretation for MD5 and SHA modes

MD5, SHA-1, SHA-256 (See Message digest hardware accelerator (MDHA) functionality)				
Code	**Interpretation**		**Code**	**Interpretation**
00h	Hash without key		03h	Reserved
01h	HMAC using a non-derived key		04h	HMAC using the Derived HMAC Key Note, using the Derived HMAC Key saves two block computations from the HMAC computations.
02h	SMAC		05h-FFh	Reserved

Table 10-49. 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 Uninstantiate	Uninstantiate the state handle.	Error

Table 10-50. PR RNG Operation commands setting

AS Value	PR = 0	PR = 1
00 Generate	Do NOT reseed prior to generating new random data. PR bit must be zero.	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 must be zero.	Reseed the state handle. PR bit is ignored.
11 Uninstantiate	Uninstantiate the state handle. PR bit must be zero.	Uninstantiate the state handle. PR bit is ignored.

Table 10-51. TST RNG OPERATION command settings

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.[1]	• 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.[2]	• If the selected state handle is in deterministic mode, reseed the state handle. • If the selected state handle is in nondeterministic mode, generate a Test error.
11 Uninstantiate	• If the selected state handle is in nondeterministic mode, uninstantiate the state handle. • If the selected state handle is in deterministic mode, generate a Test error.[3]	• If the selected state handle is in deterministic mode, uninstantiate the state handle. • If the selected state handle is in nondeterministic mode, generate a Test error.

1. 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.
2. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test reseed operation is requested 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.
3. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test uninstantiate operation is requested for 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.

10.6.17 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 specifies a protocol operation (000, 110, 111), the PROTOCOL OPERATION fields are as shown in Table 10-53. If OPTYPE specifies an algorithm operation (OPTYPE = 010: Class 1, OPTYPE = 100: Class 2), see ALGORITHM OPERATION command. If OPTYPE specifies a PKHA operation (OPTYPE = 001), see PKHA OPERATION command.

Table 10-52. PROTOCOL OPERATION Command format

31-27	26-24	23-16
CTYPE = 10000	OPTYPE = 000, 110, or 111	PROTID
15-0		
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 ID codes and information on PROTINFO encoding are shown in Table 10-54,

Table 10-53. PROTOCOL OPERATION Command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 000b, 110b or 111b : PROTOCOL OPERATION; The OPTYPE field indicates the "directionality" of the protocol as shown below. The PROTID field is interpreted as shown in the following PROTID field description table.

Table continues on the next page...

Table 10-53. PROTOCOL OPERATION Command field descriptions (continued)

Field	Description
	If OPTYPE=000b : Unidirectional protocol If OPTYPE=110b : Decapsulation protocol If OPTYPE=111b : Encapsulation protocol If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command. If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command. All others: reserved
23-16 PROTID	Protocol Identifier This field represents the specific protocol that this descriptor is implementing. See Table 10-54 for the complete description.
PROTINFO 15-0	This value is protocol-dependent.

Table 10-54. PROTID and PROTINFO field description

PROTID (hex)	Description	PROTINFO Information
0D	For OPTYPE 110 or 111: Blob	For blobs encapsulation or decapsulation, the PROTINFO field is defined in Table 10-57 and Table 10-58. For further information concerning blobs, see Blobs.
12	OPTYPE 000: (EC)DSA Verify with Private Key else Reserved	For (EC)DSA Verify using Private Key, the PROTINFO field is defined in Table 10-60. For further information, see Verifying DSA and ECDSA signatures.
14	OPTYPE 000: DH, DSA, and ECC Key Pair Generation OPTYPE 110: MPPubK generation OPTYPE 111: MPPrivK generation else Reserved	For Key Pair Generation, MPPubK and MPPrivK, the PROTINFO field is defined in Table 10-60. For further information, see Discrete-log key-pair generation
15	OPTYPE 000: (EC)DSA_Sign OPTYPE 110: MPSign else Reserved	For (EC)DSA Sign, and MPSign, the PROTINFO field is defined in Table 10-60. For further information, see Generating DSA and ECDSA signatures.
16	OPTYPE 000: (EC)DSA_Verify else Reserved	For (EC)DSA Verify, the PROTINFO field is defined in Table 10-60. For further information, see Verifying DSA and ECDSA signatures.
17	OPTYPE 000: (EC)Diffie-Hellman else Reserved	For this function the PROTINFO field is defined in Table 10-60. For further information, see Using the Diffie_Hellman function.
18	OPTYPE 000: RSA_Encrypt else Reserved	For RSA_Encrypt, the PROTINFO field is defined in Table 10-62. For further information concerning RSA Encrypt see Implementation of the RSA encrypt operation.
19	OPTYPE 000: RSA_Decrypt else Reserved	For RSA_Decrypt, the PROTINFO field is defined in Table 10-64. For further information concerning RSA Decrypt see Implementation of the RSA decrypt operation.

Table continues on the next page...

Table 10-54. PROTID and PROTINFO field description (continued)

PROTID (hex)	Description	PROTINFO Information
1A	OPTYPE 000: RSA_Finish_KeyGen else Reserved	For RSA_Finish_Keygen, the PROTINFO field is defined in Table 10-65. See RSA Finalize Key Generation (RFKG) for further information.
1E	OPTYPE 000: EC Public Key Validation else Reserved	For EC Public Key Validation use bit [0] to select F2m validation. No other PROTINFO bits are used. See Elliptic Curve Public Key Validation for details.
20	OPTYPE 000: Derived Key MD5 else Reserved	For Derived Key MD5, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.
21	OPTYPE 000: Derived Key SHA1 else Reserved	For Derived Key SHA1, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.
22	OPTYPE 000: Derived Key SHA224 else Reserved	For Derived Key SHA224, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.
23	OPTYPE 000: Derived Key SHA256 else Reserved	For Derived Key SHA256, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.
24	else Reserved	
25	else Reserved	
2F	OPTYPE 000: Derived Key ARC4 else Reserved	For Derived Key ARC4, the PROTINFO field is defined in Table 10-56. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.
60	OPTYPE 000: Derived Key MD5 with RIF else Reserved	For Derived Key MD5 with RIF, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 10-18.
61	OPTYPE 000: Derived Key SHA1 with RIF else Reserved	For Derived Key SHA1 with RIF, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 10-18.
62	OPTYPE 000: Derived Key SHA224 with RIF else Reserved	For Derived Key SHA224 with RIF, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 10-18.
63	OPTYPE 000: Derived Key SHA256 with RIF else Reserved	For Derived Key SHA256 with RIF, the PROTINFO field is defined in Table 10-55. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 10-18.
64	else Reserved	
65	else Reserved	

Table continues on the next page...

Table 10-54. PROTID and PROTINFO field description (continued)

PROTID (hex)	Description	PROTINFO Information
6F	OPTYPE 000: Derived Key ARC4 with RIF else Reserved	For Derived Key ARC4 with RIF, the PROTINFO field is defined in Table 10-56. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 10-18.
All other values reserved.		

Table 10-55. PROTINFO definition when used with derived key protocol (DKP) for HMACs

PROTINFO	Description
PROTINFO[15:14] SRC	Input Source Control 00 - IMM - negotiated key is in words immediately following the DKP Operation Command. This option can only be used with an Immediate Output Destination (OD=00). 01 - SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command. 10 - PTR - the input key is referenced by the address found immediately following the DKP Operation Command. 11 - SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command.
PROTINFO[13:12] DST	Output Destination Control 00 - IMM - the resulting Derived HMAC Key will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting Derived HMAC Key is twice the underlying hash context length. See Table 10-132 Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source. 01 - SEQ - the resulting Derived HMAC Key will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source. 10 - PTR - the resulting Derived HMAC Key will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF. 11 - SGF - the resulting Derived HMAC Key will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR.
PROTINFO[11:0] LEN	Length of the negotiated key provided to the DKP Operation command in bytes.

Table 10-56. PROTINFO definition when used with Derived Key Protocol (DKP) for ARC4

PROTINFO	Description
PROTINFO [15]	Keys Encrypted. Both the negotiated key and the derived key are encrypted.

Table continues on the next page...

Table 10-56. PROTINFO definition when used with Derived Key Protocol (DKP) for ARC4 (continued)

PROTINFO	Description
KE	0 - Neither key encrypted 1 - both negotiated and derived key are encrypted
PROTINFO [14] EKT	Encrypted Key Type 0 - ECB-encrypted key 1 - CCM-encrypted key
PROTINFO [13-12] SD	source/destination 00 - Simple pointers for source and destination -- address immediately after the DKP operation command is the memory location where the negotiated key is read and where the derived key is written. 01 - SGF pointers for source and destination -- address immediately after the DKP operation command is the memory location where a scatter/gather table resides. This scatter/gather table directs where the negotiated key is read and where the derived key is written. 10 - SEQ sequence for source and destination -- the negotiated key is read from the input frame as defined by the SEQ IN PTR command, and the derived key is written to the output frame as defined by the SEQ OUT PTR command. 11 - Reserved
PROTINFO[11:0] LEN	Length of the negotiated key provided to the DKP Operation command in bytes.

Table 10-57. PROTINFO format when used with Blob Operations

15-10	9	8	7-4	3	2	1-0
Reserved	TK	EKT	K2KR	Sec_Mem	Black_Key	Blob_ Format

Table 10-58. PROTINFO field descriptions when used with Blob Operations

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

Table continues on the next page...

Table 10-58. PROTINFO field descriptions when used with Blob Operations (continued)

Field	Description
	Used only for black blob encapsulation/decapsulation. Ignored otherwise. Specifies the encryption/decryption mode for black keys. Also used when deriving the blob key encryption key. Consequently, the same EKT setting must be used when decapsulating a black blob as was used when encapsulating that black blob. This prevents a black key being converted between AES-ECB and AES-CCM by encapsulating it as a blob and then decapsulating the blob in the other encryption mode. 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 Blob types differentiated by content) 0000 Memory 0001 Class 1 key register 0011 Class 2 key register 0111 Class 2 key register (Derived HMAC Key) 1001 PKHA E RAM 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.
1-0 Blob_Format	The format of the blob. 00 Normal Blob. The output is composed of the encrypted blob key, the encrypted data, and MAC tag. 01 Reserved 10 Master Key Verification Blob. This blob type is intended for verifying the master key and the key derivation. The master key is used for key derivation in the Trusted and Secure security states. The test key is used in the Nonsecure state. 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 a one-way function 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. 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.

Table 10-59 shows the format of the PROTINFO field for discrete log public key protocols, including:

- Key pair generation (see Discrete-log key-pair generation)

- DSA sign (see Generating DSA and ECDSA signatures)
- DSA verify (see Verifying DSA and ECDSA signatures)
- Diffie-Hellman (see Using the Diffie_Hellman function).

Table 10-60. describes the bit values of this field.

Table 10-59. PROTINFO format when used with Discrete Log Protocol

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format for Sign function	Reserved			SIGN_NO_TEQ	MES_REP		HASH			SIGN_2ND_HALF_ONLY	SIGN_1ST_HALF_ONLY	EXT_PRI	TEST	ENC_PRI	ECC/DL	F2M/Fp
Format for MPSign functions	Reserved			SIGN_NO_TEQ	MES_REP		HASH			Reserved	Reserved	EXT_PRI	TEST	ENC_PRI	ECC/DL	F2M/Fp
Format for Verify function	Reserved				MES_REP		HASH			Reserved					ECC/DL	F2M/Fp
Format for Keypair Generation functions	Reserved								KPG_IETF_DH	EKT_Z	ENC_Z	EXT_PRI	KPG_NO_TEQ	ENC_PRI	ECC/DL	F2M/Fp
Format for MP Keypair Generation functions	Reserved												KPG_NO_TEQ	Reserved		

Table 10-60. PROTINFO field descriptions when used with Discrete Log Protocol

Field	Description
SIGN_NO_TEQ	For Signature Generation (SIGN) protocol and MPSign: disable Timing Equalization during SIGN. 0 Run SIGN using normal Timing Equalization protection. 1 Run SIGN with NO Timing Equalization protection.
MES_REP	For Signature Generation (SIGN) and Verification (VERIFY) protocols, this field indicates the format of the message. 00 : F input is a message representative. 01 : Calculate the message representative from the message (using a SEQ IN PTR command), and the hash function specified by the HASH field. The message representative is calculated using the equivalent of EMSA1 (IEEE-1363). 10 : F input is a hashed message, with length specified in the PDB. Protocol will format the message as required. 11 : Reserved.
HASH	Hash function used to calculate a message representative from a message; valid when MES_REP=01.

Table continues on the next page...

Table 10-60. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

Field	Description
	000 MD5 001 SHA-1 010 SHA-224 011 SHA-256
KPG_IETF_DH	For KPG, this bit enables running IETF_style DH 0 No IETF-style DH. 1 Run KPG with IETF-style Diffie-Hellman.
PRI_VERIFY_NO_TEQ	For Signature Verification with Private Key (PRI_VERIFY) protocol only. Disable Timing Equalization protection. 0 Run PRI_VERIFY with Timing Equalization protection enabled. 1 Run PRI_VERIFY with Timing Equalization protection disabled.
SIGN_2ND_HALF_ONLY	For Signature Generation (SIGN) protocol only; otherwise reserved. Run 2nd half (signature "d" generation) only. 0 Run full SIGN or 1st half, depending on SIGN_1ST_HALF_ONLY setting. 1 Run 2nd half of SIGN only, generating 'd' result. Requires SIGN_1ST_HALF_ONLY = 0.
SIGN_1ST_HALF_ONLY	For Signature Generation (SIGN) protocol only; otherwise reserved. Run 1st half (signature "c" generation) only. 0 Run full SIGN or 2nd half, depending on SIGN_2ND_HALF_ONLY setting. 1 Run 1st half of SIGN only, generating 'c' result. Requires SIGN_2ND_HALF_ONLY = 0.
EKT_Z	if ENC_Z=1, Key Encryption type (Used only with DH; otherwise reserved.) 0 Secret output is encrypted with AES-ECB mode. 1 Secret output is encrypted with AES-CCM mode.
ENC_Z	Encrypt the DH shared secret (Used only with DH; otherwise reserved.) 0 The DH output is public and is unencrypted. 1 The DH output is secret and encrypted.
EXT_PRI	if ENC_PRI=1, Encrypted key type for private key 0 Private key is encrypted with AES-ECB mode. 1 Private key is encrypted with AES-CCM mode.
KPG_NO_TEQ	KPG_NO_TEQ For KPG, MPPrivK and MPPubK. 0 Key Pair Generation runs with Timing Equalization protection. 1 Kep Pair Generation runs with Timing Equalization disabled.
TEST	TEST

Table continues on the next page...

Table 10-60. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

Field	Description
	0 Signature generation protects the per message secret. 1 Signature generation outputs the per message secret, to aid in testing and verification. This is not allowed in trusted or secure states.
ENC_PRI	Encrypted private key 0 Private key is not encrypted. ENC_PRI must be 0 if SIGN_2ND_HALF_ONLY=1. 1 Private key must be decrypted before use (see KEY command for further information.). For Key Generation, this causes the Private Key to be encrypted.
ECC/DL	Public Key operation type 0 DL: Discrete Log 1 ECC: Elliptic Curve Cryptography
F2M/Fp	Finite Field type 0 Fp: Prime Field 1 F2M: Binary field

Table 10-61 shows the format of the PROTINFO field for the RSA encrypt protocol. Table 10-62 describes the bit values.

Table 10-61. PROTINFO format when used with RSA Encrypt Protocol

15-13	12	1-7	6-4	3-2	1-0
Reserved	FMT	Reserved	fff	Reserved	OP

Table 10-62. PROTINFO field descriptions when used with RSA Encrypt Protocol

Field	Description
15-13	Reserved.
12 FMT	Format of data 0 No formatting 1 EME-PKCS1-v1_5 encryption encoding function
11-7	Reserved.
6-4 fff	Encryption type for f 000b f is not encrypted (This is the only value permitted when OP = 00b). 001b f is to be encrypted with the JDKEK using ECB mode. 011b f is to be encrypted with the JDKEK using CCM mode. 101b f is to be encrypted with the TDKEK using ECB mode. 111b f is to be encrypted with the TDKEK using CCM mode.

Table continues on the next page...

Table 10-62. PROTINFO field descriptions when used with RSA Encrypt Protocol (continued)

Field	Description
	All other values are reserved.
3-2	Reserved.
1-0 OP	Operation. 00b Public Key n, e, f in - f is a user-supplied value (fff must be 000b) 01b Public Key n, e, f out - f is a random value (f can be encrypted on output, fff can be any non-reserved value) All other values are reserved.

Table 10-63 shows the format of the PROTINFO field for the RSA Decrypt Protocol. Table 10-64 describes the bit values.

Table 10-63. PROTINFO format when used with RSA Decrypt Protocol

15-13	12	11	10-8	7	6-4	3-2	1-0
Reserved	FMT	Reserved	ppp	Reserved	fff	Reserved	Key Form

Table 10-64. PROTINFO field descriptions when used with RSA Decrypt Protocol

Field	Description
15-13	Reserved.
12 FMT	Format of data 0 No formatting 1 EME-PKCS1-v1_5 encryption decoding function
11	Reserved.
10-8 ppp	Type of private key encryption 000 private key is not encrypted 001b private key components are each encrypted with the JDKEK using ECB mode 011b private key components are each encrypted with the JDKEK using CCM mode 101b private key components are each encrypted with the TDKEK using ECB mode 111b private key components are each encrypted with the TDKEK using CCM mode All other values are reserved.
7	Reserved.
6-4 fff	Type of encryption for f. 000b f is not to be encrypted 001b f is to be encrypted with the JDKEK using ECB mode 011b f is to be encrypted with the JDKEK using CCM mode 101b f is to be encrypted with the TDKEK using ECB mode 111b f is to be encrypted with the TDKEK using CCM mode

Table continues on the next page...

Table 10-64. PROTINFO field descriptions when used with RSA Decrypt Protocol (continued)

Field	Description
	All other values are reserved.
3	No TEQ option. Set to 1 to enable no-TEQ.
2	Reserved.
1-0 Key Form	Form of the Private Key 00b Private Key input in the form #1: n, d 01b Private Key input in the form #2: p, q, d 10b Private Key input in the form #3: p, q, dp, dq, c 11b Private Key input in the form #4: p, q, dp, dq, cr, rrp, rrq

Table 10-65. PROTINFO format when used with RSA Finish KeyGen

15-8	7	6	5	4	3	2	1-0
Reserved	FMT4	ENC_OUT	Reserved	EKT	SKIP_D	SKIP_PQ	FUNCTION

Table 10-66. PROTINFO field descriptions when used with RSA Finish Keygen Protocol

Field	Description
15-8	Reserved.
7 FMT4	Format 4 0 Generate original format outputs 1 Generate original format outputs except *c*, but also generate *cr*, *rrp*, *rrq* (If FUNCTION=10b, only *n*, *d* and *d* size are output.)
6 ENC_OUT	Encrypt Outputs 0 Do not encrypt generated private key components 1 Encrypt generated private key (ECB mode, unless EKT=1) (Note that *n* and *d* size are not encrypted.)
5 Reserved	Reserved. Must be 0.
4 EKT	Encrypted Key Type 0 Do not use CCM-encryption 1 CCM-encrypt private key components (valid only if PROTOCOL OPERATION Command's ENC bit is 1)
3 SKIP_D	Skip length check of *d* 0 Check that *d* is at least one bit longer than 1/2 of the bit length of *n*. 1 Skip length check of *d*. It is an error to set SKIP_D to 1b if FUNCTION is set to 11b.
2 SKIP_PQ	Skip check of upper 100 bits of *p* and *q* 0 Check upper 100 bits of *p* and *q* to see whether |*p*-*q*| is too small

Table continues on the next page...

Table 10-66. PROTINFO field descriptions when used with RSA Finish Keygen Protocol (continued)

Field	Description
	1 Do not check upper 100 bits of *p* and *q*
1-0 FUNCTION	Function 00 Compute all key components listed in Key Form, including *d* 01 Compute all key components listed in Key Form except *d*, which is an input 10 From *p*, *q*, *e*, compute *n*, *d* and *d* size. 11 From *p*, *q*, *c*, compute just *cr*, *rrp*, *rrq*. This can be used to compute Form 4 from a private key that is already in Form 3. It is an error to set FUNCTION to 11b if FMT4 is not set to 1.

10.6.18 PKHA OPERATION command

If OPTYPE = 001 (PKHA), the fields are as shown in Table 10-67. This OPTYPE is used to perform public key operations in the public key hardware accelerator (PKHA). All data for a PKHA operation must already be in place before the function will begin executing. Therefore, this operation does not start until all data transactions have completed and the input data FIFO is empty.

The format of the PKHA MODE field depends on which of the four types of PKHA functions the OPERATION command specifies:

- Clear memory
- Modular arithmetic
- Elliptic curve
- Copy memory

A detailed description of the PKHA MODE fields is found in Table 10-68. The OPERATION command does not complete until the PKHA is done.

When the PKHA operation completes without error, DECO clears the DONE flag and the Mode Register so another operation can be specified.

Table 10-67. PKHA OPERATION command format

31-27	26-24	23-20	19-16
CTYPE = 10000	OPTYPE = 001	ALG = 1000	PKHA_MODE_MS
15-12	11-0		
Reserved	PKHA_MODE_LS		

Table 10-68. PKHA OPERATION command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 001b :PKHA OPERATION : The PKHA_MODE fields are interpreted as shown in the following tables. If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command. If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION command.
23-20 ALG	Algorithm Set ALG=1000b. All other values are reserved.
19-16 PKHA_MODE_MS	PKHA Mode This field contains the value that will be loaded into the upper 4 bits of the PKHA Mode register. Its content depends on which of the four types of PKHA functions, clear memory, modular arithmetic function, or copy memory, is specified in the Function field (bits 5-0). The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function, PKHA OPERATION : Arithmetic Functions, PKHA OPERATION : Elliptic Curve Functions and PKHA OPERATION : copy memory functions.
15-12	Reserved
11-0 PKHA_MODE_LS	PKHA Mode This field contains the value that will be loaded into the lowest 12 bits of the PKHA Mode register. The least-significant six bits of this field is interpreted as a Function field, as shown in the row below. The format of the PKHA_MODE_MS field and the other bits of the PKHA_MODE_LS field depend on the PKHA function specified in the Function field: clear memory, modular arithmetic function, or copy memory. The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function, PKHA OPERATION : Arithmetic Functions, PKHA OPERATION : Elliptic Curve Functions, and PKHA OPERATION : copy memory functions.
5-0 *Function*	PKHA function to be performed. *(Note that the function is encoded in the least-significant six bits of the PKHA_MODE_LS field.)* If Function=000001b : Clear Memory. (See Clear Memory (CLEAR_MEMORY) function) If Function=010000b or 010001b : Copy Memory. (See PKHA OPERATION : copy memory functions) If Function=001001b, 001010b, 001011b, or 011100b : Elliptic Curve function. (See PKHA OPERATION : Elliptic Curve Functions) If Function=000010b - 001111b or 010110b .. 011111b : Modular Arithmetic function. (See PKHA OPERATION : Arithmetic Functions) All other values of the Function field are reserved.

10.6.18.1 PKHA OPERATION : clear memory function

Table 10-69. PKHA Mode register format for clear memory function

19	18	17	16		11-10	9	8	7	6	5-0

Table continues on the next page...

Table 10-69. PKHA Mode register format for clear memory function (continued)

Aram	Bram	Eram	Nram	.	Reserved	Q3	Q2	Q1	Q0	Function
PKHA_MODE_MS				.	*PKHA_MODE_LS*					

If the Function field in PKHA MODE specifies the clear memory function, PKHA expects to be in the format shown in Table 10-69. The PKHA RAMs to be cleared may be selected in any combination. Selecting one or more Quadrants for clearing will cause only the specified quadrants (of the specified RAMs) to be cleared. If no Quadrants are selected, then the whole RAM will be cleared.

Table 10-70. PKHA mode register field descriptions for clear memory function

Bits	Description
19	Aram This bit selects the A RAM for zeroization. 0: A not selected 1: A selected.
18	Bram This bit selects the B RAM for zeroization. 0: B not selected 1: B selected
17	Eram This bit selects the E RAM for zeroization. 0: E not selected 1: E selected
16	Nram This bit selects the N RAM for zeroization. 0: N not selected 1: N selected
11-10	Reserved
9	Quadrant 3 This bit selects the Quadrant 3 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
8	Quadrant 2 This bit selects the Quadrant 2 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
7	Quadrant 1 This bit selects the Quadrant 1 RAM for zeroization.

Table continues on the next page...

Table 10-70. PKHA mode register field descriptions for clear memory function (continued)

Bits	Description
	0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid is E RAM is selected.
6	Quadrant 0 This bit selects the Quadrant 0 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
5-0	Function The Function value for clearmemory is 000001.

10.6.18.2 PKHA OPERATION : Arithmetic Functions

Table 10-71. PKHA Mode Register Format for Arithmetic Functions

19	18	17	16-12	11	10	9-8	7-6	5-0
inM	outM	F2m	Reserved	Reserved	Teq	OutSel	Reserved	Function
PKHA_MODE_MS				*PKHA_MODE_LS*				

Table 10-72. PKHA Mode register, format for arithmetic operation

Bits	Description
19 inM	Inputs in Montgomery form. Indicates whether the inputs are in Montgomery form. If inM=0 : Normal value representation If inM=1 : Montgomery form. (Not valid for all functions.)
18 outM	Outputs in Montgomery form. Indicates whether the outputs are to be left in Montgomery form or converted to normal values. If outM=0 : Normal value representation If outM=1 : Montgomery form. (Not valid for all functions.)
17 F2m	F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function. If F2m=0 : Integer If F2m=1 : Binary polynomial. (Not valid for all functions.)
16-12	Reserved
11	Reserved
10 Teq	Timing Equalized. Indicates that a timing equalized version of the function should be executed. If Teq=0 : No timing equalization If Teq=1 : Timing equalization. (Not valid for all functions.)
9-8 OutSel	Output destination select. Indicates which memory should contain the output of the selected function. If OutSel=00b : B

Table continues on the next page...

Table 10-72. PKHA Mode register, format for arithmetic operation (continued)

Bits	Description
	If OutSel=01b : A If OutSel=10b : Reserved If OutSel=11b : Reserved
7-6	Reserved
5-0 Function	Function. Indicates which arithmetic function to execute. If Function=000010b : Modular Addition (A + B) mod N If Function=000011b : Modular Subtraction 1 (A - B) mod N If Function=000100b : Modular Subtraction 2 (B - A) mod N If Function=000101b : Modular Multiplication (A x B) mod N If Function=000110b : Modular Exponentiation A^E mod N If Function=000111b : Modular Reduction A mod N If Function=001000b : Modular Inversion A^{-1} mod N If Function=001100b : Montgomery Radix Constant R^2 mod N If Function=001110 : Greatest Common Divisor GCD(A,N)-see note below If Function=001111 : Miller-Rabin Primality Test -see note below If Function=010110b : Modular Simultaneous Exponentiation $A0^E$ * $A2^B$ mod N If Function=010111b : Modular Square Root If Function=011000b : Modular Double A (A + A) mod N If Function=011001b : Modular Double B (B + B) mod N If Function=011010b : Modular Square A (A x A) mod N If Function=011011b : Modular Cube A (A x A x A) mod N If Function=011101b : Shift Right A If Function=011110b : Compare A B If Function=011111b : Evaluate A All other values for this field are currently reserved or are Table 10-69,Table 10-76, or Table 10-80. **NOTE:** When using the GCD function or any ECC function, a divide-by-zero error occurs if the value of the most significant digit of N is all zeros. **NOTE:** When using the Miller-Rabin primality test function, if the most-significant digit of N is all zeros, the result is composite regardless of the value of N.

NOTE

Note that the arithmetic functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

Table 10-73. List of mode values for PKHA Integer Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
MOD_ADD	Integer Modular Addition	B	0	00002	Integer Modular Addition (MOD_ADD) function
		A	0	00102	
MOD_SUB_1	Integer modular subtraction (A - B)	B	0	00003	Integer Modular Subtraction (MOD_SUB_1) function
		A	0	00103	
MOD_SUB_2	Integer modular subtraction (B - A)	B	0	00004	Integer Modular Subtraction (MOD_SUB_2) function
		A	0	00104	
MOD_MUL MOD_MUL_TEQ	Integer modular multiplication	B	0	00005	Integer Modular Multiplication (MOD_MUL)
		A	0	00105	
	Timing equalized version	B	1	00405	
		A	1	00505	
MOD_MUL_IM MOD_MUL_IM_TEQ	Integer Modular Multiplication with Montgomery Inputs	B	0	80005	Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM)
		A	0	80105	
	Timing equalized version	B	1	80405	
		A	1	80505	
MOD_MUL_IM_OM MOD_MUL_IM_OM_TEQ	Integer Modular Multiplication with Montgomery Inputs and Outputs	B	0	C0005	Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function
		A	0	C0105	
	Timinq equalized version	B	1	C0405	
		A	1	C0505	
MOD_EXP MOD_EXP_TEQ	Integer Modular Exponentiation	B	0	00006	Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ)
		A	0	00106	
	Timing equalized version	B	1	00406	
		A	1	00506	
MOD_EXP_IM MOD_EXP_IM_TEQ	Integer Modular Exponentiation with Montgomery Inputs	B	0	80006	Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function
		A	0	80106	
	Timing equalized version	B	1	80406	
		A	1	80506	
MOD_AMODN	Integer Modular Reduction	B	0	00007	Integer Modulo Reduction (MOD_AMODN)
		A	0	00107	
MOD_INV	Integer Modular Inversion	B	0	00008	Integer Modular Inversion (MOD_INV)
		A	0	00108	
MOD_R2	Integer R^2 mod N	B	0	0000C	Integer Montgomery Factor Computation (MOD_R2)
		A	0	0010C	
MOD_GCD	Integer Greatest Common Divisor	B	0	0000E	Integer Greatest Common Divisor (MOD_GCD)
		A	0	0010E	

Table continues on the next page...

Table 10-73. List of mode values for PKHA Integer Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
PRIME_TEST	Miller_Rabin primality test	B	0	0000F	Miller_Rabin Primality Test (PRIME_TEST)
		A	0	0010F	
MOD_SML_EXP	Integer Modular Simultaneous Exponentiation	B	0	00016	Integer Simultaneous Modular Exponentiation (MOD_SML_EXP)
		A	0	00116	
MOD_SQRT	Integer Modular Square Root	B	0	00017	Integer Modular Square Root (MOD_SQRT)
		A	0	20017	
MOD_DBL_A	Integer Modular Double A	B	0	20018	
		A	0	20118	
MOD_DBL_B	Integer Modular Double B	B	0	20019	
		A	0	20119	
MOD_SQR MOD_SQR_TEQ	Integer Modular Square A	B	0	0001A	Integer Modular Square (MOD_SQR and MOD_SQR_TEQ)
		A	0	0011A	
	Timing equalized version	B	1	0041A	
		A	1	0051A	
MOD_IM_SQR MOD_IM_SQR_TEQ	Integer Modular Square A. Montgomery Input	B	0	8001A	Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ)
		A	0	8011A	
	Timing equalized version	B	1	8041A	
		A	1	8051A	
MOD_IM_OM_SQR MOD_IM_OM_SQR_TEQ	Integer Modular Square A, Montgomery Input, Montgomery Output	B	0	C001A	Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ)
		A	0	C011A	
	Timing equalized version	B	1	C041A	
		A	1	C051A	
MOD_CUBE MOD_CUBE_TEQ	Integer Modular Cube A	B	0	0001B	Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ)
		A	0	0011B	
	Timing equalized version	B	1	0041B	
		A	1	0051B	
MOD_CUBE_IM MOD_CUBE_IM_TEQ	Integer Modular Cube A, Montgomery input	B	0	8001B	Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ)
		A	0	8011B	
	Timing equalized version	B	1	8041B	
		A	1	8051B	
MOD_CUBE_IM_OM MOD_CUBE_IM_OM_TEQ	Integer Modular Cube A, Montgomery input, Montgomery output	B	0	C001B	Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ)
		A	0	C011B	

Table continues on the next page...

Table 10-73. List of mode values for PKHA Integer Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
	Timing equalized version	B	1	C041B	
		A	1	C051B	

1. PKHA_MODE_MS concatenated with 0000b concatenated with PKHA_MODE_LS

Arithmetic functions on a binary polynomials (characterestic two) (F2M). All operate in polynomial basis.

Table 10-74. List of mode values for PKHA Binary Polynomial Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
F2M_ADD	Binary Polynomial Modular Addition	B	0	20002	Binary Polynomial (F_{2^m}) Addition (F2M_ADD) function
		A	0	20102	
F2M_MUL F2M_MUL_TEQ	Binary Polynomial Modular Multiplication	B	0	20005	Binary Polynomial (F_{2^m}) Modular Multiplication (F2M_MUL)
		A	0	20105	
	Timing equalized version	B	1	20405	
		A	1	20505	
F2M_MUL_IM F2M_MUL_IM_TEQ	Binary Polynomial Modular Multiplication with Montgomery Inputs	B	0	A0005	Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function
		A	0	A0105	
	Timing equalized version"	B	1	A0405	
		A	1	A0505	
F2M_MUL_IM_OM F2M_MUL_IM_OM_TEQ	Binary Polynomial Modular Multiplication with Montgomery Inputs and Output	B	0	E0005	Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function
		A	0	E0105	
	Timing equalized version	B	1	E0405	
		A	1	E0505	
F2M_EXP F2M_EXP_TEQ	Binary Polynomial Modular Exponentiation	B	0	20006	Binary Polynomial (F_{2^m}) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)
		A	0	20106	
	Timing equalized version	B	1	20406	
		A	1	20506	
F2M_AMODN	Binary Polynomial Modular Reduction	B	0	20007	Binary Polynomial (F_{2^m}) Modulo Reduction (F2M_AMODN)
		A	0	20107	

Table continues on the next page...

Table 10-74. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
F2M_INV	Binary Polynomial Modular Inversion	B	0	20008	Binary Polynomial (F_{2^m}) Modular Inversion (F2M_INV)
		A	0	20108	
F2M_R2	Binary Polynomial R^2 mod n	B	0	2000C	Binary Polynomial (F_{2^m}) R^2 Mod N (F2M_R2) Function
		A	0	2010C	
F2M_GCD	Binary Polynomial Greatest Common Divisor	B	0	2000E	Binary Polynomial (F_{2^m}) Greatest Common Divisor (F2M_GCD) Function
		A	0	2010E	
F2M_SQR F2M_SQR_TEQ	Binary Polynomial Modular A Square	B	0	2001A	
		A	0	2011A	
	Timing equalized version	B	1	2041A	
		A	1	2051A	
F2M_IM_SQR F2M_IM_SQR_TEQ	Binary Polynomial Modular A Square. Montgomery input	B	0	A001A	
		A	0	A011A	
	Timing equalized version	B	1	A041A	
		A	1	A051A	
F2M_IM_OM_SQR F2M_IM_OM_SQR_TEQ	Binary Polynomial Modular Square A, Montgomery input, Montgomery output	B	0	E001A	
		A	0	E011A	
	Timing equalized version	B	1	E041A	
		A	1	E051A	
F2M_CUBE F2M_CUBE_TEQ	Binary Polynomial Modular Cube A	B	0	2001B	
		A	0	2011B	
	Timing equalized version	B	1	2041B	
		A	1	2051B	
F2M_CUBE_IM F2M_CUBE_IM_TEQ	Binary Polynomial Modular Cube A. Montgomery input	B	0	A001B	
		A	0	A011B	
	Timing equalized version	B	1	A041B	
		A	1	A051B	
F2M_CUBE_IM_OM F2M_CUBE_IM_OM_TEQ	Binary Polynomial Modular Cube A, Montgomery input, Montgomery output	B	0	E001B	Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ)
		A	0	E011B	
	Timing equalized version	B	1	E041B	
		A	1	E051B	

Table continues on the next page...

Table 10-74. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
F2M_SML_EXP	Binary Polynomial Modular Simultaneous Exponentiation	B	0	20016	Binary Polynomial (F_{2^m}) Simultaneous Modular Exponentiation (F2M_SML_EXP)
		A	0	20116	

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

These functions are grouped here because they do not fall into one of the previous categories of PKHA functions.

Table 10-75. List of mode values for Miscellaneous Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
Shift Right A	Right Shift	B	0	0001D	Right Shift A (R_SHIFT) function
		A	0	0011D	
Compare A B	Comparison	(no output)	0	0001E	Compare A B (COMPARE) function
Evaluate A	Compute sizes	B	0	0001F	Evaluate A (EVALUATE) function
		A	0	0011F	

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

10.6.18.3 PKHA OPERATION : copy memory functions

Table 10-76. PKHA Mode register, format for copy memory functions

19-17	16	11-10	9-8	7-6	5-0
Source Register	Destination Register		Source Segment	Destination Segment	Function
PKHA_MODE_MS	*PKHA_MODE_LS*				

Table 10-77. PKHA Mode register, field descriptions for copy memory functions

Bits	Description
19-17	Source Register. Specifies the register to be copied from.

Table continues on the next page...

Table 10-77. PKHA Mode register, field descriptions for copy memory functions (continued)

Bits	Description
Source Register	If Source Register=000 : A If Source Register=001 : B If Source Register=011 : N All other values are currently reserved.
16 Destination Register 11-10	Destination Register. Specifies the register to be copied to. If Destination Register=000 : A If Destination Register=001 : B If Destination Register=010 : E If Destination Register=011 : N All other values are currently reserved. **NOTE:** The source register and destination register fields must not be the same.
9-8 Source Segment	Source Segment. Used when copying a register segment to specify which segment in the source register to copy from. If Source Segment=00 : Segment 0 If Source Segment=01 : Segment 1 If Source Segment=10 : Segment 2 If Source Segment=11 : Segment 3 **NOTE:** These bits must be zero when E is the destination register.
7-6 Destination Segment	Destination Segment. Used when copying a register segment to specify which segment in the Destination Register to copy to. If Destination Segment=00 : Segment 0 If Destination Segment=01 : Segment 1 If Destination Segment=10 : Segment 2 If Destination Segment=11 : Segment 3 **NOTE:** These bits must be zero when E is the destination register.
5-0 Function	Function. Indicates which copy function to execute. If Function=010000 : Copy Memory N-Size (copies the same number of words as are in the modulus.) If Function=010001 : Copy Memory SRC-Size (copies the number of words specified in the source's size register)

This table gives the encodings for the PKHA memory-to-memory copy functions. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size (Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

Table 10-78. Mode values for PKHA copy memory functions

	Destination Memory			
Source Memory	**A**	**B**	**N**	**E**
A		00410 00411	00C10 00C11	00810 00811
B	20010 20011		20C10 20C11	20810 20811
N	60010 60011	60410 60411		60810 60811

This table gives the encodings for the PKHA memory-to-memory copy functions, when segments are involved. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size (Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

Table 10-79. Mode values for PKHA copy memory by segment functions

	Destination Quadrant											
Source Quadrant	**A0**	**A1**	**A2**	**A3**	**B0**	**B1**	**B2**	**B3**	**N0**	**N1**	**N2**	**N3**
A0					00410 00411	00450 00451	00490 00491	004D0 004D1	00C10 00C11	00C50 00C51	00C90 00C91	00CD0 00CD1
A1					00510 00511	00550 00551	00590 00591	005D0 005D1	00D10 00D11	00D50 00D51	00D90 00D91	00DD0 00DD1
A2					00610 00611	00650 00651	00690 00691	006D0 006D1	00E10 00E11	00E50 00E51	00E90 00E91	00ED0 00ED1
A3					00710 00711	00750 00751	00790 00791	007D0 007D1	00F10 00F11	00F50 00F51	00F90 00F91	00FD0 00FD1
B0	20010 20011	20050 20051	20090 20091	200D0 200D1					20C10 20C11	20C50 20C51	20C90 20C91	20CD0 20CD1
B1	20110 20111	20150 20151	20190 20191	201D0 201D1					20D10 20D11	20D50 20D51	20D90 20D91	20DD0 20DD1
B2	20210 20211	20250 20251	20290 20291	202D0 202D1					20E10 20E11	20E50 20E51	20E90 20E91	20ED0 20ED1
B3	20310 20311	20350 20351	20390 20391	203D0 203D1					20F10 20F11	20F50 20F51	20F90 20F91	20FD0 20FD1
N0	60010 60011	60050 60051	60090 60091	600D0 600D1	60410 60411	60450 60451	60490 60491	604D0 604D1				
N1	60110 60111	60150 60151	60190 60191	601D0 601D1	60510 60511	60550 60551	60590 60591	605D0 605D1				
N2	60210	60250	60290	602D0	60610	60650	60690	606D0				

Table continues on the next page...

Table 10-79. Mode values for PKHA copy memory by segment functions (continued)

	Destination Quadrant											
Source Quadrant	A0	A1	A2	A3	B0	B1	B2	B3	N0	N1	N2	N3
	60211	60251	60291	602D1	60611	60651	60691	606D1				
N3	60310	60350	60390	603D0	60710	60750	60790	607D0				
	60311	60351	60391	603D1	60711	60751	60791	607D1				

10.6.18.4 PKHA OPERATION : Elliptic Curve Functions

NOTE

The elliptic curve functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

Table 10-80. PKHA Mode Register Format for Elliptic Curve Functions

19	18	17	16		11	10	9-8	7-6	5-0
Reserved	Reserved	F2m	R2	. .	Reserved	Teq	OutSel	Reserved	Function
PKHA_MODE_MS				.	*PKHA_MODE_LS*				

Table 10-81. PKHA Mode register, format for elliptic curve operation

Bits	Description
17 F2m	F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function. If F2m=0 : Integer (prime) curve If F2m=1 : Binary polynomial curve. (Not valid for all curve types.)
16 R2	(R2 mod N). Indicates whether the term (R2 mod N) must be supplied as an input or will be calculated by the routine. If R2=0 : (R^2 mod N) is calculated and applied, if needed If R2=1 : (R^2 mod N) is an input. (Not valid for all functions.)
11 Reserved	Reserved
10 Teq	Timing Equalized. Indicates that a timing equalized version of the function should be executed. If Teq=0 : No timing equalization If Teq=1 : Timing equalization. (Not valid for all functions.)
9-8 OutSel	Output destination select. Indicates which memory should contain the output of the selected function. If OutSel=00b : B If OutSel=01b : A

Table continues on the next page...

Table 10-81. PKHA Mode register, format for elliptic curve operation (continued)

Bits	Description
	If OutSel=10b : Reserved If OutSel=11b : Reserved
7-6	Reserved
5-0 Function	Function. Indicates which elliptic curve function to execute. If Function=001001b : ECC Point Add (P1 + P2) If Function=001010b : ECC Point Double (P2 + P2) If Function=001011b : ECC Point Multiply (E x P1) if Function=011100b : ECC Check Point All other values for this field are currently reserved or are Table 10-69,Table 10-76, or Table 10-71.

Elliptic Curve Functions over a prime field (ECC_MOD), where prime p > 3. For a general discussion, see ECC_MOD: Point math on a standard curve over a prime field (F_p).

Table 10-82. List of mode values for Prime Field (F_p) Elliptic Curve Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
ECC_MOD_ADD	ECC prime field point add - affine coordinates	B	0	00009	ECC F_p Point Add, Affine Coordinates (ECC_MOD_ADD) Function
		A	0	00109	
ECC_MOD_ADD_R2	ECC prime field point add - affine coordinates, R2 input	B	0	10009	ECC F_p Point Add, Affine Coordinates, R^2 Mod N Input (ECC_MOD_ADD_R2) Function
		A	0	10109	
ECC_MOD_DBL	ECC prime field point double - affine coordinates	B	0	0000A	ECC F_p Point Double, Affine Coordinates (ECC_MOD_DBL) Function
		A	0	0010A	
ECC_MOD_MUL ECC_MOD_MUL_TEQ	ECC prime field point multiply - affine coordinates	B	0	0000B	ECC F_p Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function
		A	0	0010B	
	Timing equalized version of ECC prime field point multiply - affine coordinates	B	1	0040B	
		A	1	0050B	
ECC_MOD_MUL_R2 ECC_MOD_MUL_R2_TEQ	ECC prime field point multiply - affine coordinates, r2 mod n input	B	0	1000B	ECC F_p Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_MOD_MUL_R
		A	0	1010B	

Table continues on the next page...

Table 10-82. List of mode values for Prime Field (F_p) Elliptic Curve Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
	Timing equalized version of ECC prime field point multiply - affine coordinates, r2 mod n input	B	1	1040B	2 and ECC_MOD_MUL_R 2_TEQ) Function
		A	1	1050B	
ECC_MOD_CHECK_POINT	ECC Prime Field Point Validation	-	0	0001C	ECC F_p Check Point (ECC_MOD_CHECK _POINT) Function
ECC_MOD_CHECK_POINT_R2	ECC Prime Field Point Validation, R2 input	-	0	1001C	ECC F_p Check Point, R^2 Mod N Input, Affine Coordinates (ECC_MOD_CHECK _POINT_R2) Function

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

Elliptic Curve Functions over a binary field (ECC_F2M). All operate in polynomial basis. For a general discussion, see ECC_F2M: Point math on a standard curve over a binary field (F_{2^m}).

Table 10-83. List of mode values for Binary Field (F_{2^m}) Elliptic Curve Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
ECC_F2M_ADD	ECC binary field point add - affine coordinates	B	0	20009	ECC F_{2^m} Point Add, Affine Coordinates (ECC_F2M_ADD) Function
		A	0	20109	
ECC_F2M_ADD_R2	ECC binary field point add - affine coordinates, R2 input	B	0	30009	ECC F_{2^m} Point Add, Affine Coordinates, R^2 Mod N Input (ECC_F2M_ADD_R 2) Function
		A	0	30109	
ECC_F2M_DBL	ECC binary field point double - affine coordinates	B	0	2000A	ECC F_{2^m} Point Double - Affine Coordinates (ECC_F2M_DBL) Function
		A	0	2010A	
ECC_F2M_MUL ECC_F2M_MUL_TEQ	ECC binary field point multiply - affine coordinates	B	0	2000B	ECC F_{2^m} Point Multiply, Affine Coordinates
		A	0	2010B	

Table continues on the next page...

Table 10-83. List of mode values for Binary Field (F_{2^m}) Elliptic Curve Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)	Detailed description
	Timing equalized version of ECC binary field point multiply - affine coordinates	B	1	2040B	(ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function
		A	1	2050B	
ECC_F2M_MUL_R2	ECC binary field point multiply - affine coordinates, r2 mod n input	B	0	3000B	ECC F_{2^m} Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function
		A	0	3010B	
ECC_F2M_MUL_R2_TEQ	Timing equalized version of ECC binary field point multiply - affine coordinates, r2 mod n input	B	1	3040B	
		A	1	3050B	
ECC_F2M_CHECK_POINT	ECC Binary Polynomial Point Validation	-	0	A001C	ECC F_{2^m} Check Point (ECC_F2M_CHECK_POINT) Function
ECC_F2M_CHECK_POINT_R2	ECC Binary Polynomial Field Point Validation, R2 input	-	0	B001C	ECC F_{2^m} Check Point, R^2 (ECC_F2M_CHECK_POINT_R2) Function

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

10.6.19 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 keyed 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 keyed 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 keyed hash. This provides flexibility in changing the address or the immediate data specified by a command. For example, the following command may a LOAD command, which contains the command word itself followed by a pointer. These SIGNATURE types would allow the command word to be part of the keyed hash but would exclude the pointer from the calculation. 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 would allow a malicious user to shorten the length of the immediate data and insert additional commands that would not be included in the keyed 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 keyed hash. Because the length of the immediate data is included in the keyed hash, the length cannot be altered such that additional commands can substitute for a portion of the immediate data.

Table 10-84. SIGNATURE command format

31–27	26-20	19–16
CTYPE = 10010	Reserved	TYPE
15-0		

Table continues on the next page...

Table 10-84. SIGNATURE command format (continued)

Reserved
Additional words of SIGNATURE command
8 Words to hold the Signature (these are used in types 0000, 0001 and 0010 only)

Table 10-85. SIGNATURE command field descriptions

Field	Description
31-27 CTYPE	Command type IF CTYPE=10010b : Signature command
26-20	Reserved
19-16 TYPE	See Table 10-86
15-0	Reserved

Table 10-86. TYPE field description

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.	Type 0000, when executed, terminates execution of the descriptor normally.
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. Note that the rehash and update is always done whether the descriptor was modified or not. Following the rehash and update, descriptor execution terminates normally.
0010		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. This version allows the rehash and update to be skipped when no change has been made to the descriptor. If MATH_Z is 0, descriptor execution immediately terminates normally. Otherwise, descriptor execution terminates normally after the rehash and update.
1010	SIGNATURE command types 1010, 1011, and 1100 are used to include only a portion of the following command in the keyed hash calculation, omitting the remainder of the command 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	Type 1010 instructs CAAM to hash only the first 2 bytes of the next command.
1011		Type 1011 instructs CAAM to hash only the first 3 bytes of the next command.
1100		Type 1100 instructs CAAM to hash only the first 4 bytes of the next command.

Table continues on the next page...

Table 10-86. TYPE field description (continued)

Type	Meaning	Instructions
	and lengths without invalidating the signature. Note that the SIGNATURE command is, itself, included in the hash so that it cannot be added later.	
Others	Reserved	

Two types of the final SIGNATURE command, 0001 and 0010 described in the above table, will recompute and update the signature in memory. These types are used when the trusted descriptor modifies itself and the modified version is to be used thereafter. Note that it is up to the descriptor writer to ensure that the copy of the descriptor in memory is updated using a STORE command. However, this update should only be done once all other commands in the descriptor have completed successfully. That is, the penultimate command should be the STORE to update the descriptor and the final command must be the SIGNATURE command. (If the update was done earlier and an error was detected prior to the SIGNATURE command running, the trusted descriptor could never be run again since the signature won't match.) The final signature command will wait to run until all reads have completed, all write data has been taken by the DMA, and all internal moves have completed. However, the final signature command is not a Done Checkpoint, which means that it will not wait for CHAs to complete.

10.6.20 JUMP (HALT) command

The JUMP command has the following uses:

- Alters the execution order of descriptor commands
- Pauses execution until specified conditions are satisfied
- Halts the execution of the descriptor if specified conditions are satisfied

JUMP command format shows the format of the JUMP command, and Table 10-88 describes the JUMP command field definitions.

The JUMP command may or may not be a checkpoint depending on its conditions and type.

10.6.20.1 Jump type

The JUMP command has eight different types, distinguished by the value in the JUMP TYPE field. All of these types specify a tested condition and take some execution flow action if the tested condition evaluates as true, and simply continue with the next command if the tested condition is false. See Test type for an explanation of what it means for the tested condition to be true.

Four of these jump types are true conditional jumps, another two are 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 (e.g. 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).

10.6.20.1.1 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 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. For example, a jump of one goes to the next 32-bit word and a jump of two skips one 32-bit word. 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 10-8)

10.6.20.1.2 Local conditional increment/decrement jump

The local conditional increment/decrement jump is simply a local conditional jump that either increments or decrements a specified register, updates the math conditions based upon the result, and then evaluates the selected math conditions to determine whether or not the jump should be taken:

- If the tested condition is satisfied, command execution continues at a new point within the descriptor buffer.
- If the tested condition is not satisfied, the jump is not taken and execution continues with the command that follows the JUMP command.

Note that the increment and decrement jump types use a different JUMP command format than the other jump types. The four most-significant bits of the TEST CONDITION field are replaced with a SRC_DST field that specifies the register that is to be incremented or decremented. The least significant four bits of the TEST CONDITION field constitute the MATH CONDITION field, which specifies the tested conditions that are evaluated to determine whether the jump is taken or not.

Any of the legal choices for the SRC0 field of the MATH command which are also legal choices for the DEST field of the MATH command may be selected as the register to increment or decrement. Use the same value to select the register as is used for the MATH command.

10.6.20.1.3 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. Note that the target descriptor may not be a shared descriptor nor may the target descriptor have a shared descriptor. The pointer to the target 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.

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.

10.6.20.1.4 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 PKHA/Math condition bits (see the "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field in JUMP command format) 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.

NOTE

If the specified conditions evaluate as true, this command will always result in a nonzero status being returned for this job. Therefore, such a job will always appear to have encountered an error. The 8-bit error code will, as described above, be the PKHA and Math status flags rather than one of the predefined error codes.

10.6.20.1.5 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 PKHA/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.

NOTE

If the specified conditions evaluate as true, and the LOCAL OFFSET field is nonzero, this command will result in a nonzero status being returned for this job. That is, it will appear that such a job encountered an error. The 8-bit error code will, as described above, be a copy of the LOCAL OFFSET field rather than one of the predefined error codes.

NOTE

If the specified conditions evaluate as true, and the LOCAL OFFSET is zero, this command will terminate execution of the descriptor with normal status. That is, it will appear that such a job terminated normally. This is a convenient way to terminate execution in the middle of a descriptor when it can be determined that all work is done rather than having to jump to the end of the descriptor.

10.6.20.1.6 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. 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 only one return address can be saved, so subroutine calls cannot be nested. The descriptor writer is responsible for enforcing this as no error will be thrown if subroutine calls are nested.

NOTE

A built-in protocol is, in fact, also a special subroutine call. The return address is used to note where execution should resume following the execution of the built-in protocol. Therefore, while a protocol may be called from within a subroutine, a subsequent subroutine return will return to the command following the protocol command rather than the command following the subroutine call.

Each time a conditional subroutine call is taken or a built-in protocol is started, the return address is saved. That return address will be maintained until it is overwritten by another conditional subroutine call or built-in protocol. Therefore, it is possible to have one subroutine call that corresponds to multiple subroutine returns. It is also possible to match subroutine returns with calls to built-in protocols.

10.6.20.1.7 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 previously saved return address. If the tested condition is true, the subroutine return jumps to the saved return address. This address is the location immediately following the most recently executed command that updated the return address. One command that updates the return address is a conditional subroutine call in which the tested condition evaluated as true. The other command that updates the return address is a built-in protocol. If the tested condition evaluates to false, execution continues with the command following the subroutine return command.

NOTE

See the previous section, Conditional subroutine call, for important details on the use of the conditional subroutine return.

10.6.20.2 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. (Logical AND of all conditions.)
- 01—All specified test conditions are false. (Logical NOR of all conditions.)
- 10—Any specified test condition is true. (Logical OR of all conditions.)
- 11—Any specified test condition is false. (Logical NAND of all conditions.)

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 local conditional jump with offset 1 (signifying a jump to the following command) is a no-op because the next command in sequence is executed whether or not the jump is taken. This is true regardless of the TEST TYPE and TEST CONDITION settings. However, a wait condition can be specified to prevent the next command from executing until the conditions are satisfied. This is a common use case for the local conditional jump.

10.6.20.3 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 MATHand PKHA 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 wait conditions become true. All of the conditional wait bits must evaluate to true independent of the TEST TYPE specified. In other words, you can't wait for one of two conditional wait conditions to become true; you must wait for both. Once all the contional wait conditions are true, the jump or halt either occurs or not, depending upon whether all of the specified conditions are 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. Note that the CLASS bits are, in fact, conditional wait bits even though they are not used in the decision on whether to take the JUMP.

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 of the following are true:

- No input to the input data FIFO is pending (NIP).
- No entry in the information FIFO is pending. That is, the NFIFO is empty. (NIFP).

Because these are conditional wait bits, the command waits until all of the wait conditions are true before evaluating the remaining conditions. The evaluation depends upon the test conditions that are selected, the state of the selected conditions, and the value in the TEST TYPE field:

- 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.

10.6.20.4 JUMP command format

Table 10-87. JUMP command format

	31-27	26-25	24	23-20	19-18	17-16
	CTYPE = 10100	CLASS	JSL	JUMP TYPE	Reserved	TEST TYPE
Format used with all jump types *except* 0001 and 0011	15-8			7-0		
	TEST CONDITION			LOCAL OFFSET		

Table continues on the next page...

Table 10-87. JUMP command format (continued)

Format used with jump types 0001 and 0011	15-12	11-8	7-0
	SRC_DST	MATH CONDITION	LOCAL OFFSET
	Additional words of JUMP command		
	Pointer (one word); see Address pointers) (this field is present for non-local JUMPs only)		

Table 10-88. JUMP command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10100 : JUMP command
26-25 CLASS	Class Wait until specified class type CHA(s) is done before evaluating jump/halt conditions. For CLASS != 00, this makes the JUMP command a DONE checkpoint. If CLASS=00 : None If CLASS=01 : Class 1 If CLASS=10 : Class 2 If CLASS=11 : Both Class 1 and Class 2
24 JSL	Jump Select Type Selects which definition of the TEST CONDITION field to use. If JSL=0 : MATHand PKHA status conditions If JSL=1 : Various jump/halt and wait conditions (Note that JSL=1 is prohibited with jump types 0001 and 0011 and such usage will result in an error.)
23-20 JUMP TYPE	Jump Type Specifies the action taken by the JUMP Command. See Jump type for more information. If JUMP TYPE=0000 : Local conditional jump. Evaluates the specified TEST CONDITION to determine whether the local jump should be taken. If JUMP TYPE=0001 : Local conditional increment jump. Increments the register specified in SRC_DST before evaluating the specified MATH CONDITION. If JUMP TYPE=0010 : Conditional subroutine call. Evaluates the specified TEST CONDITION to determine whether the local subroutine call should be taken. If JUMP TYPE=0011 : Local conditional decrement jump. Decrements the register specified in SRC_DST before evaluating the specified MATH CONDITION. If JUMP TYPE=0100 : Non-local conditional jump. Evaluates the specified TEST CONDITION to determine whether the non-local jump should be taken. If JUMP TYPE=0110 : Conditional subroutine return. Evaluates the specified TEST CONDITION to determine whether the subroutine return should be taken. If JUMP TYPE=1000 : Conditional Halt. If the specified TEST CONDITION is true, this returns the PKHA/MATH bits as status. (see "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field) and halts descriptor execution with error status. If JUMP TYPE=1100 : Conditional Halt with user-specified status. If the specified TEST CONDITION is true, this returns the value in the LOCAL OFFSET field as status and halts descriptor execution with error status unless the LOCAL OFFSET is zero, in which case descriptor execution terminates normally.

Table continues on the next page...

Table 10-88. JUMP command field descriptions (continued)

Field	Description
	All other codes are reserved, and will generate an error.
19-18	Reserved
17-16 TEST TYPE	Test Type. This field defines how the condition code bits (see TEST CONDITION field) should be interpreted. See Test type for more information. If TEST TYPE=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. If TEST TYPE=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. If TEST TYPE=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. If TEST TYPE=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.
15-8 TEST CONDITION	Test Condition.***This 8-bit field is used with all jump types except 0001 and 0011.*** The interpretation of the TEST CONDITION field depends upon the value of the JSL field, a shown in Table 10-89. See JSL and TEST CONDITION fields for more information.
15-12 SRC_DST	Source/Destination. ***This four-bit field is used only with jump types 0001 and 0011. It replaces the most-significant four bits of the TEST CONDITION field.*** This field is used to select the register that will be incremented (jump type 0001) or decremented (jump type 0011) before the selected math condition is evaluated to determine whether the local jump will be taken. For 8-byte registers, only the least-significant 4 bytes are used. (That is, the length of the math operation is restricted to 4 bytes.) If SRC_DST=0000 : Math Register 0 If SRC_DST=0001 : Math Register 1 If SRC_DST=0010 : Math Register 2 If SRC_DST=0011 : Math Register 3 If SRC_DST=0111 : DECO Protocol Override Register If SRC_DST=1000 : Sequence In Length (SIL) If SRC_DST=1001 : Sequence Out Length (SOL) If SRC_DST=1010 : Variable Sequence In Length (VSIL) If SRC_DST=1011 : Variable Sequence Out Length (VSOL) All other values are reserved.
11-8 MATH CONDITION	Math condition. ***This four-bit field is used only with jump types 0001 and 0011. It is identical with the least-significant four bits of the TEST CONDITION field and uses the same definitions.*** This field is used to select the math conditions that will be evaluated to determine whether the local jump should be taken.

Table continues on the next page...

Table 10-88. JUMP command field descriptions (continued)

Field	Description			
	bit 11	bit 10	bit 9	bit 8
	MATH N	MATH Z	MATH C	MATH NV
	The result is negative.	The result is zero.	The operation resulted in a carry or borrow.	Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.
7-0 LOCAL OFFSET	For local jumps this field specifies the offset of the JUMP target from the JUMP command's address in the descriptor buffer. 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 is reported.			
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 if the jump is taken.			

Table 10-89. TEST CONDITION bit settings

Bit #	TEST CONDITION bits when JSL=0			TEST CONDITION bits when JSL=1		
15	PKHA IS_ZERO	For Finite Field operations the result of a PKHA operation is zero. For ECC operations, the result is a Point at Infinity.	Conditional Jump/Halt	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	PKHA GCD_1	The greatest common divisor of two numbers is 1 (that is, the two numbers are relatively prime).	Conditional Jump/Halt	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	PKHA IS_PRIME	The given number is probably prime (that is, it passes the Miller-Rabin primality test).	Conditional Jump/Halt	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

Table continues on the next page...

Table 10-89. TEST CONDITION bit settings (continued)

Bit #	TEST CONDITION bits when JSL=0			TEST CONDITION bits when JSL=1		
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 negative math flag is set.	Conditional Jump/Halt	NIP	No input pending. No external loads, whether from LOAD, FIFO LOAD, SEQ LOAD, SEQ FIFO LOAD or SEQ FIFO STORE (aux==10 or 10), are pending.	Conditional Wait
10	MATH Z	The zero math flag is set.	Conditional Jump/Halt	NIFP	No iNformation FIFO entries pending. The NFIFO is empty and no data is waiting in the C1 or C2 alignment blocks.	Conditional Wait
9	MATH C	The carry/borrow math flag is set.	Conditional Jump/Halt	NOP	No output pending. No external stores, whether from STORE, FIFO STORE, SEQ STORE, or SEQ FIFO STORE, are pending.	Conditional Wait
8	MATH NV	The NV math flag is set. This is the XOR of the sign bit and 2's complement overflow.	Conditional Jump/Halt	NCP	No context load pending. There is no data in flight toward the context registers via the internal or external DMA.	Conditional Wait

10.6.21 MATH and MATHI Commands

The MATH and MATHI commands compute simple mathematical functions of values in registers or specified via immediate data. The result can be written to a specified destination register or the result can be dropped. The commands set 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. In addition, the MC bit can be used to perform addition and subtraction of values larger than the math registers via borrow or carry.

Length must always be specified in the 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.

The MATHI command is useful when a one-byte immediate value is to be used. Since this immediate value is contained within the MATHI command word, this allows the MATHI command to be a single word rather than forcing the use of a two-word MATH command. This is useful since one-byte arguments are common. In some large descriptors, saving this one word several times can make the difference between fitting in the descriptor buffer and having to use multiple descriptors.

Table 10-90. MATH and MATHI Commands, format

	31–27	26	25	24	23–20	19–16
MATH:	CTYPE = 10101	IFB	NFU	STL	FUNCTION	SRC0
MATHI:	CTYPE = 11101	Reserved	NFU	SSEL	FUNCTION	SRC
	15–12	**11–8**			**7-4**	**3–0**
MATH:	SRC1	DEST			Reserved	LEN
MATHI:	DEST	IMM_VALUE				LEN

Table 10-91. MATH command, field descriptions (field descriptions for the MATHI command appear in the next table)

Field	Description
31-27 CTYPE	Command Type If CTYPE=10101b : MATH command (fields defined in this table) If CTYPE=11101b : MATHI command (fields defined in table Table 10-92 below)
26 IFB	Immediate Four Bytes If IFB=0 : Include full length immediate data in descriptor (length specified in LEN field) If IFB=1 : Use only four bytes of immediate data even if LEN is 8. This shortens the descriptor by one word when 1, 2, or 4-byte immediate data is to be used in an 8-byte operation. The immediate data will automatically be zero padded out to 8 bytes. This bit has no effect if the LEN is less than 8.
25 NFU	No Flag Update If NFU=0 : Math flags will be updated as appropriate. If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags.
24 STL	Stall. If STL=0 : Don't stall the execution of the MATH command. If STL=1 : Stall MATH command. Causes the MATH command to take one extra clock cycle.
23-20 FUNCTION	This field specifies which function to perform, as listed in the table below titled Table 10-93. The operands are specified in the SRC0 and SRC1 fields and the result is written to the destination specified in the DEST field.
19-16 SRC0	The SRC0 field indicates the source of operand 0.

Source of Operand 0	SRC0 Field Value
Math Register 0	0h
Math Register 1	1h
Math Register 2	2h
Math Register 3	3h
Immediate data from descriptor words following the MATH command[1]	4h
Protocol Override (DPOVRD), left-extended with 0s	7h
Sequence In Length (SIL), left-extended with 0s	8h
Sequence Out Length (SOL), left-extended with 0s	9h
Variable Sequence In Length (VSIL)	Ah

Table continues on the next page...

Table 10-91. MATH command, field descriptions (field descriptions for the MATHI command appear in the next table) (continued)

Field	Description
	Source of Operand 0 / **SRC0 Field Value** Variable Sequence Out Length (VSOL) — Bh ZERO (the value 0000 0000h) is used as operand 0 — Ch ONE (the value 0000 00001h) is used as operand 0 — Fh All other values for this field are reserved.
15-12 SRC1	The SRC1 field indicates the source of operand 1. **Source of Operand 1** / **SRC1 Field Value** Math Register 0 — 0h Math Register 1 — 1h Math Register 2 — 2h Math Register 3 — 3h Immediate data from descriptor words following the MATH command[1] — 4h Protocol Override (DPOVRD), left-extended with 0s — 7h Variable Sequence In Length (VSIL) — 8h Variable Sequence Out Length (VSOL) — 9h Input Data FIFO[2,3] — Ah Output Data FIFO[3,4] — Bh ONE (the value 0000 0001h) is used as operand 1 — Ch ZERO (the value 0000 00000) is used as operand 1 — Fh All other values for this field are reserved.
11-8 DEST	The DEST field specifies the destination for the result of the command as follows: **Destination for MATH operation result** / **DEST Field Value** Math Register 0 — 0h Math Register 1 — 1h Math Register 2 — 2h Math Register 3 — 3h Protocol Override — 7h Sequence In Length — 8h Sequence Out Length — 9h Variable Sequence In Length — Ah Variable Sequence Out Length — Bh No Destination. The result should not be written anywhere.[5] — Fh All other values for this field are reserved.

Table continues on the next page...

Table 10-91. MATH command, field descriptions (field descriptions for the MATHI command appear in the next table) (continued)

Field	Description
7-4	This field is 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. 1h : 1 byte 2h : 2 bytes 4h : 4 bytes 8h : 8 bytes 9h : 8 bytes, with word swapping performed prior to use if CAAM STATUS REGISTER[PLEND]=0 (i.e. Little-Endian). LEN=9h is equivalent to LEN=8h (no word swapping) if CAAM STATUS REGISTER[PLEND]=1 (i.e. Big-Endian).[6] All other values are reserved. **NOTE:** If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h 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 1, 2, or 4-byte immediate value (0 padded to the left) in the descriptor even though it is doing an 8-byte operation. **NOTE:** If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD.

1. If the data is less than 8 bytes, it is left-extended with 0s. If the data is less than 8 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, else an error is generated.

2. 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, because leaving it there is not a problem if the input data FIFO is reset. The input FIFO is not automatically reset between job descriptors with the same shared descriptor unless the CIF bit in the Shared Descriptor is set. The input data FIFO is always reset betwen jobs without, or with different, shared descriptors. Note that the descriptor must have already created an NFIFO entry to get data to the DECO alignment block, from which the MATH command will pop it.

3. 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.

4. The output 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, because leaving it there is not a problem if the output data FIFO is reset. The output FIFO is always cleared between descriptors whether shared or not.

5. No Destination is useful for setting flags when the actual result is not needed. An error will be generated if No Destination is selected when the FUNCTION is shift_l or shift_r.

6. An error will be generated for LEN=9h if IFB=1 or if both or neither of the operands is Immediate.

Table 10-92. MATHI command, field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=10101b : MATH command (see field definitions in table Table 10-91 above)

Table continues on the next page...

Table 10-92. MATHI command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>If CTYPE=11101b : MATHI command (fields defined in this table)</td></tr>
<tr><td>26</td><td>Reserved. Must be 0.</td></tr>
<tr><td>25
NFU</td><td>No Flag Update
If NFU=0 : Math flags will be updated as appropriate.
If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags.</td></tr>
<tr><td>24
SSEL</td><td>SSEL. Source Select. Selects the type and order of the operands to the math function: operand 0 <function> operand 1 -> destination
<table>
<tr><th>SSEL value</th><th>operand 0 specified by</th><th>math function specified by</th><th>operand 1 specified by</th><th>destination specified by</th></tr>
<tr><td>0</td><td>SRC0 for MATH command</td><td>FUNCTION</td><td>IMM_VALUE</td><td>DEST</td></tr>
<tr><td>1</td><td>IMM_VALUE</td><td>FUNCTION</td><td>SRC1 for MATH command</td><td>DEST</td></tr>
</table>
NOTE: If FUNCTION=Ah (FBYT) it is illegal to set SSEL to 1.</td></tr>
<tr><td>23-20
FUNCTION</td><td>This field specifies which function to perform, as listed in the table below titled Table 10-93. The operands are specified in the SRC field and the IMM_VALUE field, and the result is written to the destination specified in the DEST field.</td></tr>
<tr><td>19-16
SRC</td><td>The SRC field indicates the source of one of the operands. The SRC field has two definitions selected via the SSEL field:
If SSEL=0: the SRC field is defined the same as the MATH command's SRC0 field (see SRC0) except that IMM (4h) is not supported and will result in an error.
If SSEL=1: the SRC field is defined the same as the MATH command's SRC1 field (see SRC1) except that IMM (4h) is not supported and will result in an error.</td></tr>
<tr><td>15-12
DEST</td><td>The destination for the result of the math operation. The MATHI command DEST field is defined the same as the MATH command's DEST field (see DEST), but is shifted to the left 4 bits to make room for the IMM_VALUE field.</td></tr>
<tr><td>11-4
IMM_VALUE</td><td>The IMM_VALUE field contains an 8-bit immediate value that is left-extended with 0s. This is used as either operand 0 or operand 1, as specified in the SSEL field.</td></tr>
<tr><td>3-0
LEN</td><td>LEN denotes the length, in bytes, of the operation (and the amount by which the IMM_VALUE is left-extended with 0s).
1h : 1 byte
2h : 2 bytes
4h : 4 bytes
8h : 8 bytes
All other values are reserved.
NOTE: If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h may yield unexpected results.
NOTE: If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD.</td></tr>
</table>

Table 10-93. FUNCTION field values

Value	Type	Description	Result
0h	add	Perform addition operation on operand 0 and operand 1.	operand 0 + operand 1
1h	add_w_carry	Perform addition with a carry bit operation on operand 0 and operand 1.	operand 0 + operand 1 + MC
2h	sub	Perform subtraction operation on operand 0 and operand 1.	operand 0 - operand 1
3h	sub_w_borrow	Perform subtraction with borrow operation on operand 0 and operand 1.	operand 0 - operand 1 - MC
4h	or	Perform bitwise OR operation on operand 0 and operand 1	operand 0 | operand 1
5h	and	Perform bitwise AND operation on operand 0 and operand 1	operand 0 & operand 1
6h	xor	Perform bitwise XOR operation on operand 0 and operand 1	operand 0 ^ operand 1
7h	shift_l	Perform shift left operation. operand 0 should be shifted left by operand 1 bits; can't be used with "No Destination"	operand 0 << operand 1
8h	shift_r	Perform shift right operation. operand 0 should be shifted right by operand 1 bits; can't be used with "No Destination"	operand 0 >> operand 1
9h (MATH command only)	shld	Perform 32-bit left shift of DEST and concatenate with left 32 bits of operand 1. shld is only meaningful when DEST specifies Math Registers 0-3. For all other destinations, this function will work like an ADD with operand 0 set to 0. (That is, operand 1 will be placed into DEST.) Note that if operand 1 and DEST are the same Math Register, then shld would do a word swap. Function type shld is prohibited for the MATHI command.	{DEST[31:0], operand 1[63:32]}
Ah	zbyt or fbyt	MATH command: zbyt. Find zero bytes in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is all zeros. MATHI command: fbyt. Find the immediate byte in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is equal to IMM_VALUE. For the fbyt function it is illegal to set SSEL=1.	result is shown at left
Bh (MATH command only)	swap_bytes	Swap the order of the four bytes in the ms half of operand 0, and independently swap the order of the four bytes in the ls half of operand 0. operand 0[39:32], operand 0[47:40], operand 0[55:48], operand 0[63:56], operand 0[7:0], operand 0[15:8], operand 0[23:16], operand 0[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. Function swap_bytes is prohibited for the MATHI command.	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 Destination and then doing a JUMP based on the CZ and/or CN flags.			

All MATH and MATHI 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 operand 1.

- 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 operand that are 1. The shifter can shift any power-of-2 number of bit positions in one cycle, and there are up to two additional cycles of overhead. If an 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 operand 0 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, operand 0 or operand 1, 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.

10.6.22 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 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 or to the original pointer if RTO=1 or to the original output sequence pointer if SOP=1.
- 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). If rewinding, the LENGTH or EXT_LENGTH field is added to the current length.

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 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. Additional length may also be added via the MATH and MATHI commands.

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 10-94. SEQ IN PTR command, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTYPE = 11110					Reserved	INL	SGF	PRE	EXT	RTO	RJD	SOP	CTRL	Reserved	Reserved
15–0															
LENGTH (used if EXT = 0)															
Optional words of SEQ IN PTR command:															
Pointer (one word; see Address pointers) **NOTE:** This pointer is omitted if PRE=1 or RTO=1															
EXT_LENGTH (present if EXT = 1) (one word) **NOTE:** This field is omitted if EXT=0															

Table 10-95. SEQ IN PTR command, field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=11110b : SEQ IN PTR command
26	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. If INL=0 : No in-line descriptor If INL=1 : In-line descriptor present. 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.

Table continues on the next page...

Table 10-95. SEQ IN PTR command, field descriptions (continued)

Field	Description
	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 Using in-line descriptors for more information.
24 SGF	Scatter/Gather Table Flag. If SGF=0 : Pointer points to actual data. If SGF=1 : Pointer points to a scatter/gather table. SGF is ignored if RTO=1.
23 PRE	Previous. Add more length to the previously specified length of the input sequence. If PRE=0 : The sequence pointer is set to the value of the Pointer and the input sequence length is set to the value specified in the LENGTH or EXT LENGTH field. If PRE=1 : Command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current input sequence length. It is an error for the PRE bit and the RTO bit to both be set.
22 EXT	Extended Length If EXT=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. If EXT=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. If RTO=0 : Do not restore. If RTO=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. SGF is ignored if RTO=1.
20 RJD	Replacement Job Descriptor If RJD=0 : Don't replace job descriptor If RJD=1 : Replace job descriptor. If there is no shared descriptor, and CTRL=0, this is synonymous with the INL bit (that is, 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 Using replacement job descriptors for more information. It is an error if both RJD = 1 and INL = 1. **NOTE:** See the description below for the CTRL bit to understand how that bit can modify the above behavior.
19 SOP	Sequence Out Pointer If SOP=0 : This bit has no effect. If SOP=1 : Start a new input sequence using the pointer and SGF bit used when the current output sequence was defined. (If there was no previous sequence, behavior is undefined.) The length used is the length that has already been written to the current output sequence. This functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. By using the SOP bit, the SEQ IN PTR command allows the second pass to reference the results of the first pass. It is an error to assert SOP if PRE, EXT or RTO are set. SGF and LENGTH are ignored.
18 CTRL	CTRL. This bit is used in conjunction with the RJD bit to differentiate between a normal RJD and a control RJD. See Using replacement job descriptors for more information. If CTRL=0 and RJD=0 : This bit has no effect.

Table continues on the next page...

Table 10-95. SEQ IN PTR command, field descriptions (continued)

Field	Description
	If CTRL=0 and RJD=1 : The new descriptor is the next data to be read from the input frame. If CTRL=1 and RJD=0 : An error will be thrown. If CTRL=1 and RJD=1 : The new descriptor is found following the shared descriptor in memory. If there is no shared descriptor, an error will be thrown.
17	Reserved
16	Reserved
15-0 LENGTH	LENGTH. This is the length of the input frame. If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the input sequence. The Extended Length word is omitted. If EXT = 1 : The number of bytes in (or to be added to) the input sequence is specified in the Extended Length field. The LENGTH field is ignored.
Optional words of SEQ IN PTR command:	
31-0 POINTER	Pointer **Note, if PRE = 1, RTO = 1, or SOP = 1, this field is omitted.** If PRE = 0, Pointer specifies the starting address for an Input Sequence. See Address pointers.
31-0 EXT_LENGTH	Extended Length Field **Note, if EXT = 0, this field is omitted.** If EXT = 1, EXT_LENGTH specifies the number of bytes in (or to be added to) the Input Sequence.

10.6.23 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 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 or to the original pointer if rewinding.
- 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 rewinding, the LENGTH or EXT_LENGTH field is added to the current length if REW = 10b and is ignored if REW = 11b.

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. Additional length may also be added via the MATH and MATHI commands.

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 using the REW field. The SEQ OUT PTR command does not include a Pointer field in this case.

Table 10-96. SEQ OUT PTR command, format

	31–27	26	25	24	23	22	21-20	19	18-17	16
	CTYPE = 11111	Reserved		SGF	PRE	EXT	REW	EWS	Reserved	Reserved
	15–0									
	LENGTH (used if EXT = 0)									
Additional words of SEQ OUT PTR command:										
This pointer is omitted if PRE=1 or if rewinding	Pointer (one word; see Address pointers)									
This word is omitted if EXT=0	EXT LENGTH (used if EXT = 1)									

Table 10-97. SEQ OUT PTR command, field descriptions

Field	Description
31-27 CTYPE	Command Type. If CTYPE=11111b : SEQ OUT PTR command
26-25	Reserved
24 SGF	If SGF=0 : Pointer points to actual data. If SGF=1 : Pointer points to a scatter/gather table.
23 PRE	Previous. Add more length to the previously specified length of the Output Sequence. If PRE=0 : The sequence pointer is set to the value of the pointer and the Output Sequence Length is set to the value specified in the LENGTH or EXT LENGTH field. If PRE=1 : The SEQ OUT PTR command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current Output Sequence Length. Note that it is an error if PRE = 1 and REW = 10b or 11b.

Table continues on the next page...

Table 10-97. SEQ OUT PTR command, field descriptions (continued)

Field	Description	
22 EXT	Extended Length If EXT=0 : The 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. If EXT=1 : The output data length value is in the 32-bit EXT LENGTH field following the pointer. The 16-bit LENGTH field is ignored.	
21-20 REW	Rewind. Used to rewind an Output Sequence. If REW = 00b : Do not rewind. If REW = 01b : Error If REW = 10b : Rewind. 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. This allows returning to the beginning of a sequence to reprocess some or all of the data. DECO automatically disables the counting of bytes written to the output frame. In order to re-enable counting, use a write to the DECO CTRL Register. If REW = 11b : Rewind and Reset. The same as 10b, except that any length provided is ignored, the current output frame length is added back to the SOL (sequence output length) register and the tracking length [1] of bytes written to the output frame is reset to 0. Care must be taken if the descriptor has modified the SOL register other than as a result of decrements caused by SEQ STORE and SEQ FIFO STORE commands. Since the number of bytes written to the output frame has been reset, counting such bytes remains enabled in this case. The REW = 10b or 11b functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. Executing the SEQ OUT PTR command with REW = 11b allows the second pass to start from the beginning of the output frame as if this were the original output stream. That way the final status reported back contains the correct length.	
19 EWS	Enable Write Safe. When this bit is set, write-safe bus transactions are permitted for this output sequence. See AXI master (DMA) interface.	
18-16	Reserved	
15-0 LENGTH	If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the output sequence. If EXT = 1 : The LENGTH field is ignored.	
Optional words of SEQ OUT PTR command:		
POINTER	Pointer. Specifies the starting address for an Output Sequence. See Address pointers.	If PRE = 1 or REW != 00b, this field is omitted.
One word EXT LENGTH	If EXT = 0 : The EXT LENGTH field is omitted. If EXT = 1 : The EXT LENGTH field specifies the number of bytes in (or to be added to) the output sequence.	If EXT = 0, this field is omitted.

1. DECO tracks how many bytes have been written to the output frame so that this number can be part of the status reported when a job completes.

10.7 Protocol acceleration

CAAM is designed to accelerate the cryptographic operations associated with various network protocols. These cryptographic operations can be implemented using the descriptor commands described in sections KEY commands through SEQ OUT PTR command, but CAAM also implements specialized descriptor commands for particular networking protocols. Each such command performs a sequence of operations that are equivalent to a series of the more general descriptor commands; for example, all the protocols in this section manage the input data and output FIFOs directly -- a SEQ FIFO LOAD command is required in a descriptor only if there is data in the input frame that is not to be handled by the protocol. These protocols often require that state information (for example, sequence numbers) be maintained per security association.

The specialized protocol commands implemented by CAAM use data structures called protocol data blocks (PDBs) embedded within the descriptor to specify protocol options and hold state information. Typically these protocol commands and their associated PDBs are contained in shared descriptors, so that the same protocol options and state information can be shared among all the Job Descriptors that identify the PDUs within a particular security association. The PDB is embedded within the shared descriptor immediately following the header, and the START INDEX field in the header is used to skip over the PDB to continue executing the commands within the shared descriptor.

If the protocol requires that state information be updated, CAAM writes the updated information back to the PDB in the shared descriptor located in system memory.

Sharing is described in Shared descriptors. Sharing significantly impacts Protocol operation in particular, because CAAM protocols tend to use a shared descriptor PDB to share state across many jobs within a flow. One example is a sequence or packet number -- it is important that only one packet be encapsulated with a given sequence number. Therefore sharing type as described in Table 10-8 is to be carefully considered when crafting a descriptor.

CAAM protocols maintain a lock called OK to Share in order to allow for wait sharing. For example, consider IPsec encapsulation using CBC mode. CBC requires every packet include an Initialization Vector (an IV). For IPsec, either the IV can be the final block of ciphertext from the previous packet (a Chained IV), or it can be a random value (a Random IV). The IPsec protocol state machine will block WAIT sharing of a shared descriptor until a Chained IV has been prepared and OK to Share is signalled. If instead a Random IV is used, OK to Share can be set as soon as the IPsec protocol state machine has updated Sequence Number in the PDB. It is probably not useful to use WAIT sharing with a Chained IV; two jobs from a single flow can only be present in multiple DECOs for a very limited period of time.

Never and Always Sharing should be used with extreme care. Selecting share type of Always will cause a shared descriptor to be shared between DECOs without consideration of state. In the IPsec encapsulation example, Always sharing can result in packets with duplicate sequence numbers. Duplicate sequence numbers can also result from using Never sharing, as a DECO will get a fresh copy of a Shared Descriptor from system memory, without any consideration for any pending writes to update the Shared Descriptor PDB from another DECO.

CAAM includes built-in descriptor programming shortcuts for the following functions:

- Cryptographic Blob Encapsulation and Decapsulation

Each detailed description of the function includes color-coded diagrams. Figure 10-10 shows the color coding key. Note that in the diagrams, processing order is reflected top-to-bottom, and PDU content is reflected left-to-right.

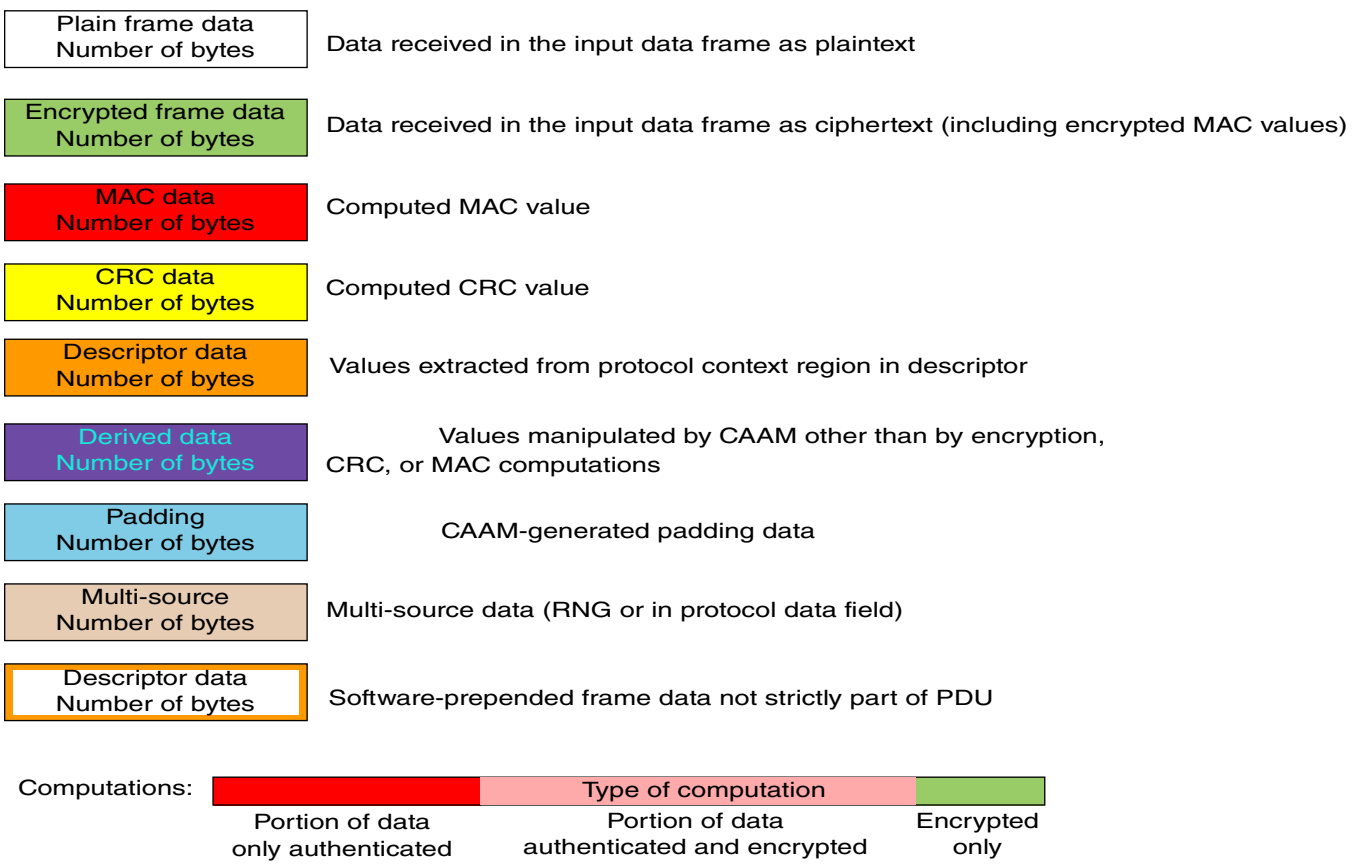


Figure 10-10. Protocol diagram color-coding key

10.8 Public Key Cryptography Operations

CAAM implements, through protocol commands, a number of public (and private) key functions. These are:

- DSA and ECDSA sign / DSA and ECDSA verify
- Diffie-Hellman (DH) and ECDH key agreement
- ECC key generation (for ECDH, ECDSA, etc.)
- ECC public key validation
- DLC key generation (for DH, DSA)
- RSA public-key and private-key primitives, for use with RSA encryption/decryption and RSA signature generation/verification
- RSA key-generation finalization, once suitable primes have been found.

In addition to public (and private) key protocol commands, CAAM also contains a Public-Key Hardware Accelerator (PKHA) that can be programmed directly for such calculations.

10.8.1 Conformance considerations

The DSA and ECDSA key-generation, signing, verification, and Diffie-Hellman functions described are intended to conform to the following specifications (except where noted). For more information refer to the NIST Cryptographic Algorithm Validation Program (CAVP) Certifications whitepaper, www.nxp.com/security, or consult these standards:

- FIPS PUB 186-4, *Digital Signature Standard (DSS)*, July 2013
- NIST SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012
- IEEE1363-2000, *IEEE Standard Specifications for Public-Key Cryptography*, January 30, 2000
- ANSI X9.42-2003, *Public Key Cryptography for the Financial Services Industry, Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*, November 19, 2003
- ANSI X9.63-2001, *Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography*, November 20, 2001
- ANSI X9.62-2005, *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 16, 2005

The notation used is from IEEE 1363-2000 because only that document provides a set of variable names and definitions consistent between both DSA and ECDSA.

Private keys for DSA and ECDSA, (as well as per-message secrets), are generated using the method of extra random bits, equivalent to that described in FIPS 186-4, (Appendix B.1.1). In B.1.1, *c* is a string of random bits, 64 bits longer than requested.

Then $x = (c \bmod (q - 1)) + 1$

CAAM uses the following equivalent version.

$x = c \bmod q$; if $(x = 0)$, choose another c

In both cases, x is uniformly distributed in the range [1, q-1].

Binary (aka Characteristic 2 or F_{2^m}) Elliptic Curves inputs and outputs are in polynomial basis and in affine (x, y) coordinates.

Assurances for the validity of all domain parameters and public keys must be obtained before invoking any of these functions. These functions assume that all domain parameters and public keys are valid and are associated with each other.

10.8.2 Specifying the ECC domain curves for the discrete-log functions

When executing an ECC function, the ECC domain curve must be specified. If the PD (Predefined Domain) bit in the function's PDB is 0, the curve parameters are supplied via the PDB. But when PD is 1, the ECDSEL (Elliptic Curve Domain Selection) field in the PDB is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in the table below.

The following variable definitions apply to the following table. Variable names (*q, r, b, c*) follow the conventions of IEEE Std 1363.

Name

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org

- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

q

This is the *field-defining* value for the elliptic curve. For F_p curves, it is the prime number used as the modulus for all point arithmetic; it is named *p* in some other publications. For F_{2^m} curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

L

This is the number of bytes needed to hold *q* and each of its associated values:, *a*,*b*,*c*, the point coordinates *x* and *y*, the result of an ECDH key agreement, etc.

r

This is the (usually prime) number which is the order of *G*, the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named *n* in some other publications.

N

This is the number of bytes needed to hold *r* and each of its associated values: private keys, each of the two components of an ECDSA signature, etc.

a

This variable, along with *q* and *b*, define the elliptic curve. For Fp, *a* is the coefficient for the *x* term. For F_{2^m}, it is the coefficient for the x^2 term.

b / c (b')

b is the coefficient for the x^0 (ones) term in an F_{2^m} elliptic curve equation. Its relationship with *c* is $b = c^4$. *c* is sometimes referred to as *b'* in NXP documentation.

Table 10-98. ECDSEL field values for built-in ECC domains

When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size.	
Value	**Name(s)**
ECC F_p domains	
00h	P-192, secp192r1, ansix9p192r1, prime192v1, ECPRGF192Random
01h	P-224, secp224r1, ansix9p224r1, wtls12, ECPRGF224Random
02h	P-256, secp256r1, ansix9p256r1, prime256v1, ECDSA-256, ecp_group_19, ECPRGF256Random

Table continues on the next page...

Table 10-98. ECDSEL field values for built-in ECC domains (continued)

When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size.	
Value	**Name(s)**
ECC F_p domains	
03h	P-384, secp384r1, ansix9p384r1, ECDSA-384, ecp_group_20, ECPRGF384Random
04h	P-521, secp521r1, ansix9p521r1, ECDSA-521, ecp_group_21, ECPRGF521Random
05h	brainpoolP160r1
06h	brainpoolP160t1
07h	brainpoolP192r1
08h	brainpoolP192t1
09h	brainpoolP224r1
0Ah	brainpoolP224t1
0Bh	brainpoolP256r1
0Ch	brainpoolP256t1
0Dh	brainpoolP320r1
0Eh	brainpoolP320t1
0Fh	brainpoolP384r1
10h	brainpoolP384t1
11h	brainpoolP512r1
12h	brainpoolP512t1
13h	prime192v2
14h	prime192v3
15h	prime239v1
16h	prime239v2
17h	prime239v3
18h	secp112r1, wtls6
19h	wtls8
1Ah	wtls9
1Bh	secp160k1, ansix9p160k1
1Ch	secp160r1, ansix9p160r1, wtls7
1Dh	secp160r2, ansix9p160r2
1Eh	secp192k1, ansix9p192k1
1Fh	secp224k1, ansix9p224k1
20h	secp256k1, ansix9p256k1
ECC F_{2^m} domains	
40h	B-163, ansix9t163r2, sect163r2, EC2NGF163Random
41h	B-233, sect233r1, ansix9t233r1, EC2NGF233Random, wtls11
42h	B-283, sect283r1, ansix9t283r1, EC2NGF283Random
43h	B-409, sect409r1, ansix9t409r1, EC2NGF409Random
44h	B-571, sect571r1, ansix9t571r1, EC2NGF571Random

Table continues on the next page...

Table 10-98. ECDSEL field values for built-in ECC domains (continued)

When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size.	
Value	**Name(s)**
ECC F_p domains	
45h	K-163, ansix9t163k1, sect163k1, EC2NGF163Koblitz, wtls3
46h	K-233, sect233k1, ansix9t233k1, EC2NGF233Koblitz, wtls10
47h	K-283, sect283k1, ansix9t283k1, EC2NGF283Koblitz
48h	K-409, sect409k1, ansix9t409k1, EC2NGF409Koblitz
49h	K-571, sect571k1, ansix9t571k1, EC2NGF571Koblitz
4Ah	wtls1
4Bh	sect113r1, wtls4
4Ch	c2pnb163v1, wtls5
4Dh	c2pnb163v2
4Eh	c2pnb163v3
4Fh	sect163r1, ansix9t163r1
50h	sect193r1, ansix9t193r1
51h	sect193r2, ansix9t193r2
52h	sect239k1, ansix9t239k1
53h	Oakley 3, ec2n_group_3
54h	Oakley 4, ec2n_group_4

10.8.3 Discrete-log key-pair generation

Some important characteristics and requirements of discrete-log key-pair generation are as follows:

- DL KEY PAIR GEN is used to generate public key-pairs. There are four variations to generate either prime field or binary field keys for either DSA or ECDSA.
- Each of the public key functions writes out the private key, followed by the public key.
- DL KEY PAIR GEN requires the parameters listed in the following table. Note that $G_{x,y}$ and $W_{x,y}$ are pointers to input buffers containing both an x and y coordinate. The two coordinates must be the same length.
- There are two parameter lengths, size of the field (L), and size of the group or private key modulus (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that the size of the buffers for $G_{x,y}$, $W_{x,y}$ and a,b must be twice L, as each holds two values of size L.

Table 10-99. Public key-generation parameters

Parameter	Input/output	Length	Definition
q	Input	L	Prime number or irreducible polynomial that creates the field
r	Input	N	Order of the field of private keys or modulus for creating private keys
a,b	Input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.)
g or $G_{x,y}$	Input	L or 2L	Generator or generator point (ECC)
s	Output	N	Private key
w or $W_{x,y}$	Output	L or 2L	Public key

10.8.3.1 Inputs to the discrete-log key-pair generation function

- For DSA, the domain parameters *q*, *r*, and *g*
- For ECDSA, the domain parameters *q*, *a, b or b',r*, and $G_{x,y}$

10.8.3.2 Assumptions of the discrete-log key-pair generation function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC bit of the Protocol Command register is set, s is treated as an encrypted key and is encrypted before being written out. When generating an encrypted key, the buffer must be large enough to hold the black key, i.e., the encrypted version of the key.

10.8.3.3 Outputs from the discrete-log key-pair generation function

- The signer's private key *s*
- For DSA, the signer's public key w
- For ECDSA, the signer's public key $W_{x,y}$

10.8.3.4 Operation of the discrete-log key-pair generation function

- Generate a private key *s*, in the range $1 \le s < r$. (Generate a random number *k*, 64 bits larger than *r*, and find $s = k \bmod r$. If $s = 0$, generate a new *k*.)
- Compute $w = g^s \bmod q$, or $W_{x,y} = sG_{x,y}$.
- Output (*s*, *w*) or (*s*, $W_{x,y}$) as the private and public keys.

10.8.3.5 Notes associated with the discrete-log key-pair generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} mod\ q$ must be given, rather than b.

For IETF DH involving domains like MODP Groups 5, 14, 15, and 16, there is no published r value. However, a value is necessary for this function, as it is the modulus used to create the private key, where 1 (< private_key < mod). The value of N should be determined based upon the desired strength of the private key; there are recommendations in the IETF RFCs and elsewhere. Both r and the private key will be N bytes long. A typical value for r would be a string containing N bytes of FFh.

When the PD (Predefined Domain) bit in the PDB is 0, the ECC curve parameters are supplied via the PDB. But when PD is 0, the ECDSEL (Elliptic Curve Domain Selection) field in the PDB is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 10-98.

Table 10-100. Public-key generation protocol data block, PDB word 1, formats with PD=0 and PD=1

PDB word 1 when PD=0	SGF (bits 31..26)	PD (=**0**) (bit 25)	reserved (bits 24..17)	L (bits 16..7)	N (bits 6..0)
PDB word 1 when PD=1	SGF (bits 31..26)	PD (=**1**) (bit 25)	reserved (bits 24..14)	ECDSEL (bits 13..7)	reserved (bits 6..0)

Table 10-101. Public-key generation protocol data block, pointers in PDB words 2, 3 ...

pointers	Discrete Log Diffie-Hellman (PD is always 0)	Elliptic Curve Diffie-Hellman (PD=*0* format)	Elliptic Curve Diffie-Hellman (PD=*1* format)
PDB pointer 1	Pointer to *q* (SGF in bit 31)	Pointer to *q* (SGF in bit 31)	Pointer to *s* (SGF in bit 28)
PDB pointer 2	Pointer to *r* (SGF in bit 30)	Pointer to *r* (SGF in bit 30)	Pointer to *Wx,y* (SGF in bit 27)
PDB pointer 3	Pointer to *g* (SGF in bit 29)	Pointer to *Gx,y* (SGF in bit 29)	
PDB pointer 4	Pointer to *s* (SGF in bit 28)	Pointer to *s* (SGF in bit 28)	
PDB pointer 5	Pointer to *w* (SGF in bit 27)	Pointer to *Wx,y* (SGF in bit 27)	
PDB pointer 6		Pointer to *a,b* (SGF in bit 26)	

10.8.4 Using the Diffie_Hellman function

Diffie-Hellman is used in key exchange and key agreement schemes. Because the output of Diffie-Hellman is a secret value, it is advisable to store the output in encrypted form and CAAM's Diffie-Hellman protocol provides this option.

Diffie-Hellman is defined for both discrete log (DH) and elliptic-curve (ECDH) forms. CAAM provides acceleration support for both forms.

10.8.4.1 Diffie_Hellman requirements

Diffie-Hellman requires the parameters listed in this table.

Table 10-102. Required Diffie-Hellman parameters

Parameter	Input/Output	Length	Definition
L	input	10 bits	Number of bytes of the the field
N	input	7 bits	Number of bytes of the private key
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	-	Unused for Diffie-Hellman
a,b	input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.)
w' or $W'_{x,y}$	input	L (DH) or 2L (ECDH)	Other party's public key
s	input	N	Own private key
z	output	L	Shared secret value
NOTE: Wx,y is a pointer to an input buffer containing both an x and a y coordinate. The two coordinates must be the same length. There are two parameter lengths, size of the field (L), and the size of the private key (N). These represent the size of the buffers, in bytes, required to hold the input and output data. The size of the buffers for $G_{x,y}$, $W_{x,y}$, and a,b must be twice L, as each holds two values of size L.			

10.8.4.2 Inputs to the Diffie-Hellman function

- For discrete logs, the domain parameters q, s (own private key), and w' (other's public key).
- For elliptic curve, the domain parameters q, s (own private key), and $W'_{x,y}$ (other's public key), a and b (or b').

Note that the domain parameters r and g (or $G_{x,y}$) are not used.

10.8.4.3 Assumptions of the Diffie-Hellman function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC_PRI bit of the Protocol Information register is set, s is treated as an encrypted key and is decrypted after being read. If the ENC_PUB bit of the protocol information is set, then z is encrypted before being written.

10.8.4.4 Outputs from the Diffie-Hellman function

The shared secret value z

10.8.4.5 Operation of the Diffie-Hellman function

- Read in the private key pointed to by s.
- For DL, compute $z = w^s \bmod q$.
- For ECC, compute new_point = s * W, and output z = x coordinate of new_point
- Output z as the shared secret.

10.8.4.6 Notes associated with the Diffie-Hellman function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given rather than b. For a detailed explanation, see ECC_F2M: Point math on a standard curve over a binary field (F_{2^m})

When the PD (Predefined Domain) bit in the PDB is 0, the ECC curve parameters are supplied via the PDB. But when PD is 0, the the ECDSEL (Elliptic Curve Domain Selection) field in the PDB is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 10-98.

Table 10-103. Diffie-Hellman protocol data block, PDB word 1, formats with PD=0 and PD=1

PDB word 1 when PD=0	SGF (bits 31..26)	PD (=**0**) (bit 25)	reserved (bits 24..17)	L (bits 16..7)	N (bits 6..0)
PDB word 1 when PD=1	SGF (bits 31..26)	PD (=**1**) (bit 25)	reserved (bits 24..14)	ECDSEL (bits 13..7)	reserved (bits 6..0)

Diffie-Hellman protocol data block, pointers in PDB words 2, 3 ...

Table 10-104. Diffie-Hellman protocol data block, PDB words 2, 3, ..

pointer or constant #	Discrete Log Diffie-Hellman (PD is always 0)	Elliptic Curve Diffie-Hellman (PD=*0* format)	Elliptic Curve Diffie-Hellman (PD=*1* format)
1	Pointer to *q* (SGF in bit 31)	Pointer to *q* (SGF in bit 31)	Pointer to *Wx,y* (SGF in bit 29)
2	(unused)	(unused)	Pointer to *s* (SGF in bit 28)
3	Pointer to *w* (SGF in bit 29)	Pointer to *Wx,y* (SGF in bit 29)	
4	Pointer to *s* (SGF in bit 28)	Pointer to *s* (SGF in bit 28)	
5	Pointer to *z* (SGF in bit 27)	Pointer to *z* (SGF in bit 27)	
6		Pointer to *a,b* (SGF in bit 26)	

10.8.5 Generating DSA and ECDSA signatures

DSA_SIGN is CAAM's hardware implementation of NIST's DSA (Digital Signature Algorithm) and ECDSA digital signing functions. It supports DSA and ECDSA in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MES_REP bit in the OPERATION Command register.

There are two parameter lengths: size of the field (L), and size of the group (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that the size of the buffers for $G_{x,y}$ and *a,b* must be twice L, as each holds two values of size L.

This table lists the DSA and ECDSA sign protocol parameters.

Table 10-105. DSA and ECDSA sign parameters

Parameter	Input/Output	Length	Definition
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	N	Order of the field of private keys
a, b	input	2L	ECC curve parameters. For binary field curves, b' rather than b is given. (ECC only.)
g or $G_{x,y}$	input	L (DSA), 2L (ECDSA)	Generator or generator point (ECC)
s	input	N	Private key
f (or m)	input	N	Message representative (typically the hash of the message) or the actual message
c	output	N	First part of digital signature
d	output	N	Second part of digital signature. The buffer for *d* must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding.

Table continues on the next page...

Table 10-105. DSA and ECDSA sign parameters (continued)

Parameter	Input/Output	Length	Definition
u	output	N	Per message random number, only in TEST mode

10.8.5.1 Inputs to the DSA and ECDSA signature generation function

- For DSA, the domain parameters q, r, and g associated with key s.
- For ECDSA, the domain parameters q, r, g, a and b associated with key s.
- The signer's private key s.
- The message representative, which is an integer $f \geq 0$, or the message itself (which is hashed to form a message representative).

10.8.5.2 Assumptions of the DSA and ECDSA signature generation function

- The private key s is in the range $1 \leq s < r$, and the domain parameters are valid and are associated with each other, (that is, parameter validation must be done prior to using this function).
- The message representative, f, is generated using an approved hashing function of the appropriate security strength.
- If the ENC bit of the Protocol Command is set, then s is treated as an encrypted key, and is decrypted before it is used.

10.8.5.3 Outputs from the DSA and ECDSA signature generation function

When running the full signature operation, the output is a pair of integers (c, d), where $1 \leq c < r$ and $1 \leq d < r$

When just the first part of the signature is generated the output is the integer c, as above, and the encrypted version of the (inverted) temporary key used in the creation of the signature. It is stored at d, and the memory there must have space for the ECB-encrypted version (i.e, rounded up to the nearest 16 bytes).

When just the second part of the signature is generated the output is d, as in the output of the complete signature operation.

10.8.5.4 Operation of the DSA and ECDSA signature generation function

- Generate a per message private key u, in the range $1 \le u < r$. (Generate a random number k, 64 bits larger than r, and find $u = k \bmod r$. If $u = 0$, generate a new k.)
- Compute $c = (g^u \bmod q) \bmod r$, or $V_{x,y} = uG_{x,y}$, $c = V_x \bmod r$. If $c = 0$, try again with a new u.
- Compute $d = u^{-1}(f + sc) \bmod r$. If $d = 0$, try again with a new u.
- Output (c, d) as the signature.
- If the TEST bit of the Protocol Command is set, then also output u. This test mode is not accessible in the Trusted or Secure states.

10.8.5.5 Notes associated with the DSA and ECDSA Signature Generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given, rather than b.

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Generation function and pointers to those arguments. Each pointer occupies one word of the PDB.

When the PD (Predefined Domain) bit in the PDB is 0, the ECC curve parameters are supplied via the PDB. But when PD is 0, the ECDSEL (Elliptic Curve Domain Selection) field in the PDB is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 10-98.

Table 10-106. DSA and ECDSA Signature Generation protocol data block, PDB word 1, formats with PD=0 and PD=1

PDB word 1 when PD=0	SGF (bits 31..26)	PD (=**0**) (bit 25)	reserved (bits 24..17)	L (bits 16..7)	N (bits 6..0)
PDB word 1 when PD=1	SGF (bits 31..26)	PD (=**1**) (bit 25)	reserved (bits 24..14)	ECDSEL (bits 13..7)	reserved (bits 6..0)

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 10-98. Note that if PD=1 for a DSA operation, a PDB error will be generated.

Table 10-107. DSA and ECDSA (PD=0) Signature Generation protocol data block, pointers in PDB words 2, 3 ...

	DSA (PD is always 0)or ECDSA when PD=0		
	There are three different commands that can be used when generating a DSA or ECDSA signature. The "Full Sign" command performs the entire DSA or ECDSA signature operation. The "First-Half Sign" command performs the first half of the signature operation, which does not require as input the message that is to be signed. The "Second-Half Sign" command takes the output of the "First-Half Sign" command and the message representative, and completes the signature operation.		
pointer or constant #	**Full Sign**	**First-Half Sign**	**Second-Half Sign**
1	Pointer to *q* (SGF in bit 31)	Pointer to *q* (SGF in bit 31)	Pointer to *r* (SGF in bit 30)
2	Pointer to *r* (SGF in bit 30)	Pointer to *r* (SGF in bit 30)	Pointer to *s* (SGF in bit 28)
3	Pointer to *g* (DSA) or *Gx,y* (ECDSA) (SGF in bit 29)	Pointer to *g* (DSA) or *Gx,y* (ECDSA) (SGF in bit 29)	Pointer to *f* (MR=0) or *m* (MR=1) (SGF in bit 27)
4	Pointer to *s* (SGF in bit 28)	Pointer to *c* (SGF in bit 26)	Pointer to *c* (SGF in bit 26)
5	Pointer to *f* (MR=0) or *m* (MR=1) (SGF in bit 27)	Pointer to *d* (SGF in bit 25)	Pointer to *d* (SGF in bit 25)
6	Pointer to *c* (SGF in bit 26)	Pointer to *a,b* (SGF in bit 22)*included if **ECDSA**, else this pointer word is omitted from the PDB*	*ml* (One 32-bit word, no SGF) *Included if MR=1, else this word is omitted from PDB*
7	Pointer to *d* (SGF in bit 25)	Pointer to *u* (SGF in bit 22)*included only if TEST=1*	Pointer to *u* (SGF in bit 22)*included only if TEST=1*
8	*ml* (One 32-bit word, no SGF)*Included only if MR=1*		
9	Pointer to *u* (SGF in bit 22)*included only if TEST=1*		

Table 10-108. ECDSA (PD=1) Signature Generation protocol data block, pointers in PDB words 2, 3 ...

	ECDSA when PD=1 (curve parameters are supplied by hardware, selected via the ECDSEL field)	
	There are three different commands that can be used when generating an ECDSA signature. The "Full Sign" command performs the entire ECDSA signature operation. The "First-Half Sign" command performs the first half of the signature operation, which does not require as input the message that is to be signed. The "Second-Half Sign" command takes the output of the "First-Half Sign" command and the message representative, and completes the signature operation.	
pointer or constant #	**Full Sign or Second-Half Sign**	**First-Half Sign**
1	Pointer to *s* (SGF in bit 28)	Pointer to *c* (SGF in bit 26)
2	Pointer to *f* (MR=0) or *m* (MR=1) (SGF in bit 27)	Pointer to *d* (SGF in bit 25)
3	Pointer to *c* (SGF in bit 26)	Pointer to *u* (SGF in bit 22)*included only if TEST=1*
4	Pointer to *d* (SGF in bit 25)	
5	*ml* (One 32-bit word, no SGF)*Included if MR=1, else this word is omitted from the PDB*	
6	Pointer to *u* (SGF in bit 22)*included only if TEST=1*	

10.8.6 Verifying DSA and ECDSA signatures

DSA_VERIFY is the digital signature algorithm (DSA) verification function. It supports both DSA and ECDSA, in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MES_REP bit in the OPERATION command.

There are two parameter lengths:

- Size of the field (L)
- Size of the subgroup (N)

These are given in bytes, and denote the size of the buffer required to hold each parameter. Note that the size of the buffers for $G_{x,y}$, $W_{x,y}$ and *a,b* must be twice L, as each holds two values of size L.

Table 10-109. DSA and ECDSA Verify parameters

Parameter	Input/Output	Length (bytes)	Definition
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	N	Order of the subgroup of private keys
a, b	input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECDSA only.)
g or $G_{x,y}$	input	L (DSA), 2L (ECDSA)	Generator or generator point (ECDSA)
w or $W_{x,y}$	input	L (DSA), 2L (ECDSA)	Public key
f (or m)	input	N	Message representative (typically the hash of the message) or the actual message
c	input	N	First part of digital signature
d	input	N	Second part of digital signature
Temp	input/output	L (DSA) 2L (ECDSA)	Temporary storage for intermediate results

10.8.6.1 Inputs to the DSA and ECDSA signature verification function

- For DSA, the domain parameters *q*, *r*, and *g* associated with key *w*
- For ECDSA, the domain parameters *q*, *r*, $G_{x,y}$, *a* and *b* associated with key $W_{x,y}$
- For signature verification using the public key, the signer's public key *w* (DSA) or $W_{x,y}$ (ECDSA)

- The received message representative, which, if MR=0, is the message representative, an integer $f \geq 0$, or, if MR=1, is the message itself (which is hashed to form a message representative)
- The received signature to be verified, which is a pair of integers (c, d)

10.8.6.2 Assumptions of the DSA and ECDSA signature verification function

- The public key (w or $W_{x,y}$) and the domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- The message representative, f, is generated using an approved hashing function of the appropriate security strength.

10.8.6.3 Outputs from the DSA and ECDSA signature verification function

- If the signature is correct, this function terminates normally.
- If the signature is not correct, this function terminates with an error code.

10.8.6.4 Operation of the DSA and ECDSA signature verification function

- Check that c is in the range [1, r-1]. If not, terminate with error code invalid signature.
- Check that d is in the range [1, r-1]. If not, terminate with error code invalid signature.
- For DSA, compute $c' = ((G^{d^{-1}f} \bmod q)(w^{d^{-1}c} \bmod q)) \bmod r$.
- For ECDSA, compute $P_{x,y} = d^{-1}fG_{x,y} + d^{-1}cW_{x,y}$, and then if $P_{x,y}$ is the point at infinity, terminate with error code invalid signature, else let $c' = P_x \bmod r$.
- If $c' \neq c$, then terminate with error code invalid signature.
- Continue as valid.

10.8.6.5 Notes associated with the DSA and ECDSA Signature Verification function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given, rather than b.

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Verification function and pointers to those arguments. Each pointer occupies one word of the PDB.

Parameter information is as follows:

- L is the number of bytes in various data buffers. L the size of the prime number or irreducible polynomial representing the cryptographic field.
- N is another length, in number of bytes in data buffers. N is the size of the number representing the order of the subgroup of private keys within the field.
- All parameters are pointers to data buffers of size L or N, (or 2L for elliptic curve points and a,b).

The protocol data block for DSA is shorter than for ECDSA, as the pointer to a,b is absent.

A temporary buffer is required during the verification of the signature.

- For DSA, the temporary buffer must be at least L bytes.
- For ECDSA, the temporary buffer must be at least 2L bytes.

The ENC bit of PROTOINFO is ignored for signature verification because only public keys are used. If the MES_REP bit of the OPERATION command is set to 1, the pointer to f (or m) points to the message to be signed, rather than to a message representative. The message length field of the protocol data block (only used when the MES_REP bit of the OPERATION command is set to 1) defines the length of the message to be signed.

When the PD (Predefined Domain) bit in the PDB is 0, the ECC curve parameters are supplied via the PDB. But when PD is 0, the ECDSEL (Elliptic Curve Domain Selection) field in the PDB is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 10-98.

Table 10-110. DSA and ECDSA Signature Verification protocol data block, PDB word 1, formats with PD=0 and PD=1

PDB word 1 when PD=0	SGF (bits 31..26)	PD (=**0**) (bit 25)	reserved (bits 24..19)	L (bits 18..7)	N (bits 6..0)
PDB word 1 when PD=1	SGF (bits 31..26)	PD (=**1**) (bit 25)	reserved (bits 24..14)	ECDSEL (bits 13..7)	reserved (bits 6..0)

Table 10-111. DSA or ECDSA (PD=0) Signature Verification protocol data block, pointers in PDB words 2, 3 ...

	DSA or ECDSA when PD=0
pointer or constant #	There are two versions of the DSA or ECDSA Signature Verification commands. One version takes the public key of the signer as an input. The other version takes the private key of the signer as an input. Verifying with the private key is a faster operation, so the latter version is useful for verifying self-signed data that may have been tampered with.
1	Pointer to *q* (SGF in bit 31)
2	Pointer to *r* (SGF in bit 30)
3	Pointer to *g* (DSA) or *Gx,y* (ECDSA) (SGF in bit 29)
4	***Verification with Public Key:*** Pointer to *w* (DSA) or *Wx,y* (ECDSA) (SGF in bit 28) ***Verification with Private Key:*** Pointer to *s* (SGF in bit 28)
5	Pointer to *f* (MR=0) or *m* (MR=1) (SGF in bit 27)
6	Pointer to *c* (SGF in bit 26)
7	Pointer to *d* (SGF in bit 25)
8	Pointer to Temp (SGF in bit 24)
9	*ml* (One 32-bit word, no SGF)*Included only if MR=1*

Table 10-112. ECDSA (PD=1) Signature Verification protocol data block, pointers in PDB words 2, 3 ...

	ECDSA when PD=1
pointer or constant #	There are two versions of the ECDSA Signature Verification command. One version takes the public key of the signer as an input. The other version takes the private key of the signer as an input. Verifying with the private key is a faster operation, so the latter version is useful for verifying self-signed data that may have been tampered with.
1	***Verification with Public Key:*** Pointer to *Wx,y* (SGF in bit 28) ***Verification with Private Key:*** Pointer to *s* (SGF in bit 28)
2	Pointer to *f* (MR=0) or *m* (MR=1) (SGF in bit 27)
3	Pointer to *c* (SGF in bit 26)
4	Pointer to *d* (SGF in bit 25)
5	Pointer to Temp (SGF in bit 24) *Included if Verification with Private Key, otherwise this word is omitted from the PDB*
6	*ml* (One 32-bit word, no SGF)*Included only if MR=1*

10.8.7 Elliptic Curve Public Key Validation

The Elliptic Curve Public Key Validation protocol checks whether a public key is properly part of an ECC Domain. This is done by first checking whether the point is valid (that is, a solution to the curve), and then, if the "co-factor" k is not one, performing a point multiplication of the key with r and checking that the result is proper. The result of

these checks is pass (job status OK. It will generate a 'protocol key error' if it is not. It can also generate a 'protocol command error' or a 'protocol pdb error' should there be a problem with Descriptor encoding.

Table 10-113. Elliptic Curve Public key-validation protocol parameters

Parameter	Input/output	Length	Definition
q	Input	L	Prime number or irreducible polynomial that creates the field
r	Input	N	Order of the field of private keys or modulus for creating private keys
a,b	Input	2L	ECC curve parameters. For binary field curves, b' rather than b is used.
$W_{x,y}$	Input	2L	Public key
k	Input	N	Co-factor of the key order to the curve-order (F_p only)

10.8.7.1 Inputs to the Elliptic Curve public key validation function

- The domain parameters *q*, *a, b or b'*, $W_{x,y}$, *r*, and *k*.

10.8.7.2 Outputs from the Elliptic Curve public key validation function

The Elliptic Curve public key validation function does not produce any outputs and returns Job Status as follows:

- Status OK: public key is part of the ECC Domain. Also, if co-factor k > 1, then the point multiplication rW is also part of the ECC Domain.
- Status 8A: public key or rW are not part of the ECC Domain.
- Status 81: protocol command error - invalid protocol command in descriptor.
- Status 82: protocol pdb error - non-sensical pdb in descriptor.

10.8.7.3 Operation of the Elliptic Curve public key validation function

- Verify W is part of the ECC Domain
- Verify rW is part of the ECC Domain (only if co-factor k > 1)

10.8.7.4 Notes associated with the Elliptic Curve public key validation function

Co-factor k is 1 for common F_p curves.

Table 10-114. Elliptic Curve public key validation protocol data block, PDB word 1, formats with PD=0 and PD=1

PDB word 1 when PD=0	SGF (bits 31..28)	PD (=**0**) (bit 27)	reserved (bits 26..17)	L (bits 16..7)	N (bits 6..0)
PDB word 1 when PD=1	SGF (bits 31..28)	PD (=**1**) (bit 27)	reserved (bits 26..14)	ECDSEL (bits 13..7)	reserved (bits 6..0)

Table 10-115. Elliptic Curve public key validation protocol data block, PDB words 2, 3 ...

pointers / word	Elliptic Curve Diffie-Hellman (PD=*0* format)	Elliptic Curve Diffie-Hellman (PD=*1* format)
PDB pointer 1	Pointer to *q* (SGF in bit 31)	Pointer to $W_{x,y}$ (SGF in bit 28)
PDB pointer 2	Pointer to *r* (SGF in bit 30)	
PDB pointer 3	Pointer to *a, b (b')* (SGF in bit 29)	
PDB pointer 4	Pointer to $W_{x,y}$ (SGF in bit 28)	
PDB Co-factor word	Co-factor k	

10.8.8 RSA Finalize Key Generation (RFKG)

CAAM is able to complete RSA key generation given primes p and q and the public exponent e or, to complete the creation of Form 4, given primes p and q and private exponent d. The primes p and q must have the same byte length (#p). CAAM can be configured to compute, or skip computation of, the remaining elements of the public and private key.

The computations performed are:

- n = p*q
- d = 1/e mod LCM(p-1, q-1)
- dp = d mod (p-1)
- dq = d mod (q-1)
- c = 1/q mod p

it will also

- check p and q to determine whether they are 'too close' (per FIPS 186-3). This will occur if they are long enough. It will not be effective if they are not of the same bit length (that is, high order bits of p and q are not the same).
- Compute #d if d is being computed

- Check that #d > #p.
- Check that #n and the computed #n are the same

For Form 4, it will compute

- rrp = RnRp (see MOD_RR)
- rrq = RnRq (see MOD_RR)
- cr c in Montgomery (for modulus p)

RSA Finalize Key Generation PDB

Table 10-116. RSA Finalize Key Generation PDB when FUNCTION != 11b

Descriptor header (one or two words)			
SGF (11 bits)	Reserved (21 bits)		
Reserved (23 bits)			#p (9 bits)
Rsv (6 bits)	#n (10 bits)	Reserved (6 bits)	#e (10 bits)
Reference to p			
Reference to q			
Reference to e			
Reference to n			
Reference to d			
Reference to #d			
Reference to dp (not required if FUNCTION=10b)			
Reference to dq (not required if FUNCTION=10b)			
FMT4=0b Reference to c; FMT4=1b Reference to cr (not required if FUNCTION=10b)			
Reference to rrp (only required if FMT4=1b)			
Reference to rrq (only required if FMT4=1b)			

The fields #d and #n contain right-aligned 10-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. Note that the size of d must be at least as large as the size of an encrypted n.

A reference is a pointer, either to the data or to a scatter-gather table. The pointer is one or two words long, depend upon the platform. All outputs (other than n and d) will be #p, or, if encrypting the key, as many bytes as it would take to hold the encrypted value.

The references to dp, dq, and crr may be omitted if those values are not to be generated. The references to cr, rrp, and rrq may be omitted if those values are not to be generated.

This figure shows the format of the SGF field.

Table 10-117. RSA Finalize Key Generation PDB - SGF field when FUNCTION != 11b

31	30	29	28	27	26	25	24	23	22	21
ref p	ref q	ref e	ref n	ref d	ref #d	ref dp	ref dq	FMT4=0b ref c FMT4=1b ref cr	ref rrp	ref rrq
If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.										

The following tables provide the PDB and s/g entries when FUNCTION=11b.

Table 10-118. RSA Finalize Key Generation PDB when FUNCTION = 11b

Descriptor header (one or two words)		
SGF (6 bits	Reserved (26 bits)	
Reserved (23 bits)		#p (9 bits)
Reserved (6 bits)	#n (10 bits)	Reserved (16 bits)
Reference to p		
Reference to q		
Reference to c		
Reference to cr		
Reference to rrp		
Reference to rrq		

Table 10-119. RSA Finalize Key Generation PDB - SGF field when FUNCTION = 11b

31	30	29	28	27	26
ref p	ref q	ref c	ref cr	ref rrp	ref rrq
If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.					

10.8.9 Implementation of the RSA encrypt operation

CAAM implements an RSA encrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. It is the "RSA public key primitive" and it is commonly used to encrypt a secret or to verify a signature.

When used for signature verification, it is actually "unscrambling" the signature so that its contents may be verified. The input must be passed "raw" to the RSA function.

CAAM implements the RSA encrypt operation in the following form:

g = RSA-Encrypt(n, e, FORMAT, #f, f, fff)

The variables have the following definitions:

- n, e represent the public key
- Before the RSA math is performed, FORMAT specifies the format to be used for encoding f (none or PKCS #1 v1.5 encryption)
- f is the value to be RSA-encrypted (input value; will be output value if random data ('f out')is selected)
- #f is the size in bytes of f
- fff represents the type of encryption applied to f if it is output by CAAM
- g is the RSA-encrypted value of (the possibly formatted version of) f

The RSA Encrypt function is implemented with the OPERATION command. See PROTOCOL OPERATION Commands for details on selecting this operation. See Table 10-61 and Table 10-62 for details about the PROTINFO field in the OPERATION command.

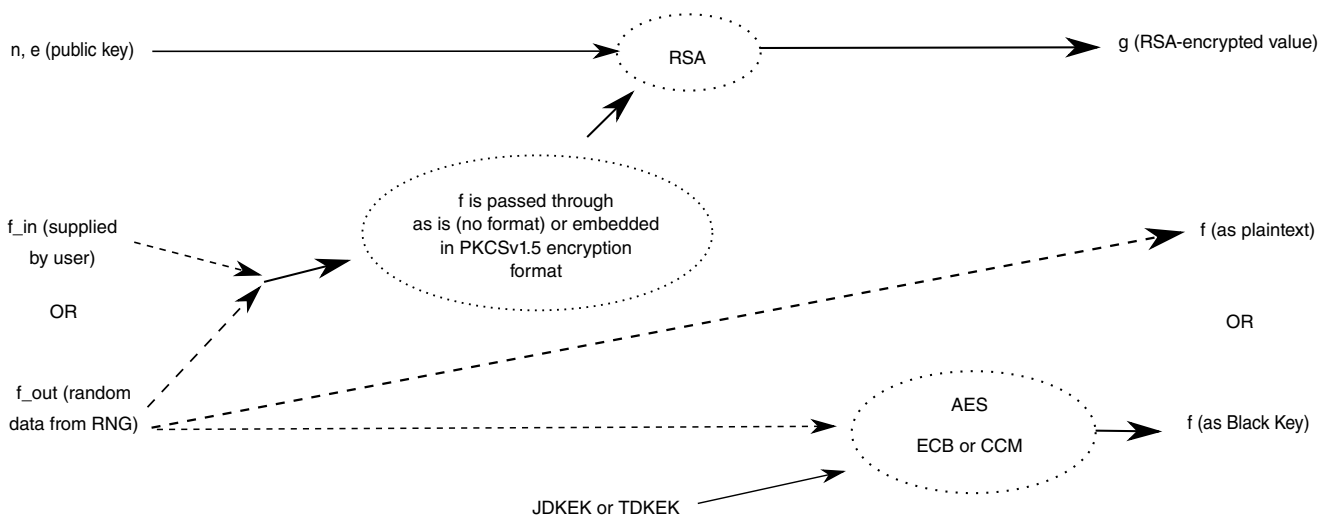


Figure 10-11. RSA encrypt operation

The user may either supply a plaintext value to be RSA encrypted (f_in) or may opt to have CAAM generate #f bytes of random data from the RNG (f_out). The latter option allows f to be stored encrypted as a black key.

Once the value of f is known and possibly wrapped in PKCSv1.5 encoding, it is RSA-encrypted and the result stored as g.

The PDB for the RSA encrypt operation is shown below.

Table 10-120. RSA Encrypt PDB

SGF (4 bits)	Rsv (4 bits)	#e (12 bits)	#n (12 bits)
Reference to f			
Reference to g			
Reference to n			
Reference to e			
Reserved (20 bits)			#f (12 bits)

All references are either 32-bit, 36-bit or 40-bit address pointers.

The fields #e, #n and #f contain right-aligned 12-bit values that indicate the size of the e, n and f inputs, respectively. The format of the SGF field is shown below.

Table 10-121. RSA Encrypt PDB; SGF field

31	30	29	28
ref f	ref g	ref n	ref e
If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.			

10.8.10 Implementation of the RSA decrypt operation

CAAM implements an RSA decrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. This is the "RSA private key primitive" and it is commonly used either to decrypt a secret or to create a signature (sign a message).

When used for signing a message, it is actually "scrambling" the signature; the output must be allowed to pass "raw" from the RSA function.

CAAM implements the RSA decrypt operation in the form:

f = RSA-Decrypt((private key), FORMAT, g, fff)

The variables have the following definitions:

- (private key) represents the private key, in one of four forms
- After the RSA math is performed on g, FORMAT specifies the format to be used for decoding f (none or PKCS #1 v1.5 encryption).
- g is the input value
- fff represents the type of encryption applied to f when it is output by CAAM
- f is the RSA-decrypted output value.

This operation leaves #f (size of the plaintext f, in bytes) in the MATH0 register. This may be important if using a FORMAT of PKCS #1 v1.5 encryption.

The RSA Decrypt function is implemented via the OPERATION Command. See PROTOCOL OPERATION Commands for details on selecting this operation. See Table 10-63 and Table 10-63 for details concerning the PROTINFO field in the OPERATION Command.

As the private key is an input and is considered sensitive, it may be supplied in Black Key form. The components, individually, of the private key would then be decrypted using the appropriate key encryption key and cryptographic mode prior to use. Note that n is never encrypted. CAAM allows the private key input to be provided in any of four different forms that enable increasing efficiency of computation:

- #1 (n, d)
- #2 (p, q, d)
- #3 (p, q, dp, dq, c)
- #4 (p, q, dp, dq, cR, RRp, and RRq) See RSA Finalize Key Generation to compute Form 4.

The RSA Decrypt function is implemented via the OPERATION Command. See PROTOCOL OPERATION Commands for details on selecting this operation. See Table 10-63 and Table 10-64 for details concerning the PROTINFO field in the OPERATION Command.

The operation of RSA decrypt when using form #1, in which the private key is input as (n, d), is illustrated below.

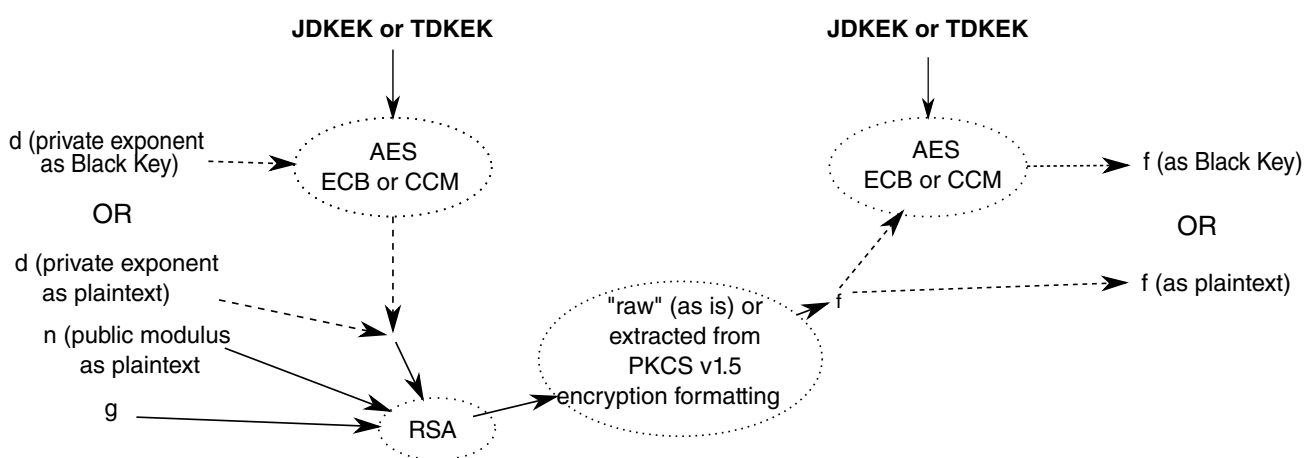


Figure 10-12. RSA decrypt operation - private key form #1

The PDB for private key form #1 is shown below. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 10-122. RSA decrypt PDB - private key form #1

SGF (4 bits)	Rsv (4 bits)	#d (12 bits)	#n (12 bits)
Reference to g			
Reference to f			
Reference to n			
Reference to d			

The fields #d and #n contain right-aligned 12-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. This figure shows the format of the SGF field.

Table 10-123. RSA decrypt PDB - private key form #1; SGF field

31	30	29	28
ref g	ref f	ref n	ref d
If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.			

The RSA decrypt operation also accepts the private key in the form (p, q, d). This form (form #2) of the RSA decrypt operation is illustrated in this figure.

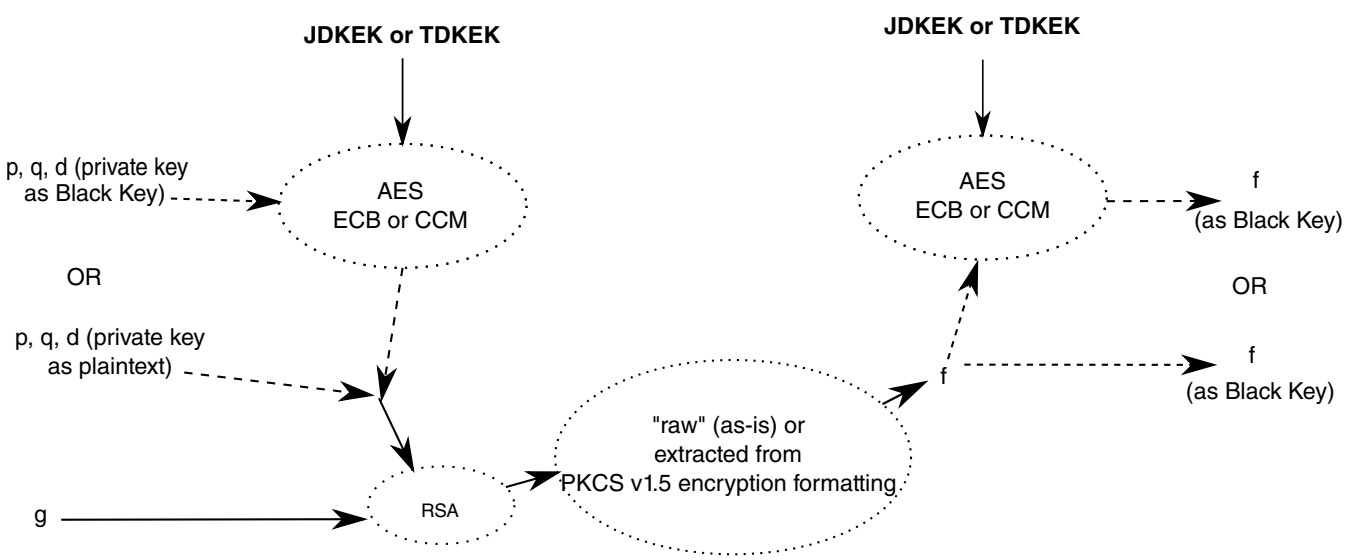


Figure 10-13. RSA decrypt operation - private key form #2

This figure shows the PDB for private key form #2. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 10-124. RSA decrypt PDB - private key form #2

SGF (7 bits)	Rsv (1 bit)	#d (12 bits)	#n (12 bits)
Reference to g			
Reference to f			
Reference to d			
Reference to p			
Reference to q			
Reference to tmp*p*			
Reference to tmp*q*			
Reserved (8 bits)		#q (12 bits)	#p (12 bits)

The fields #d, #n, #q and #p contain right-aligned 12-bit values that indicate the size (plaintext sizes, in bytes) of d, n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. tmp*p* needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp*q* needs to be as long as q (either #q, or. if q is encrypted, as big as the encrypted value of q). This figure shows the format of the SGF field.

Table 10-125. RSA decrypt PDB - private key form #2; SGF field

31	30	29	28	27	26	25
ref g	ref f	ref d	ref p	ref q	ref tmp*p*	ref tmp*q*
If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.						

The RSA decrypt operation also accepts the private key in form #3, (p, q, dp, dq, c). dp, dq, and c are

- $dp = d \bmod p\text{-}1$
- $dq = d \bmod q\text{-}1$
- $c = q^{-1} \bmod p$

The operation of this form of RSA decrypt is illustrated in this figure.

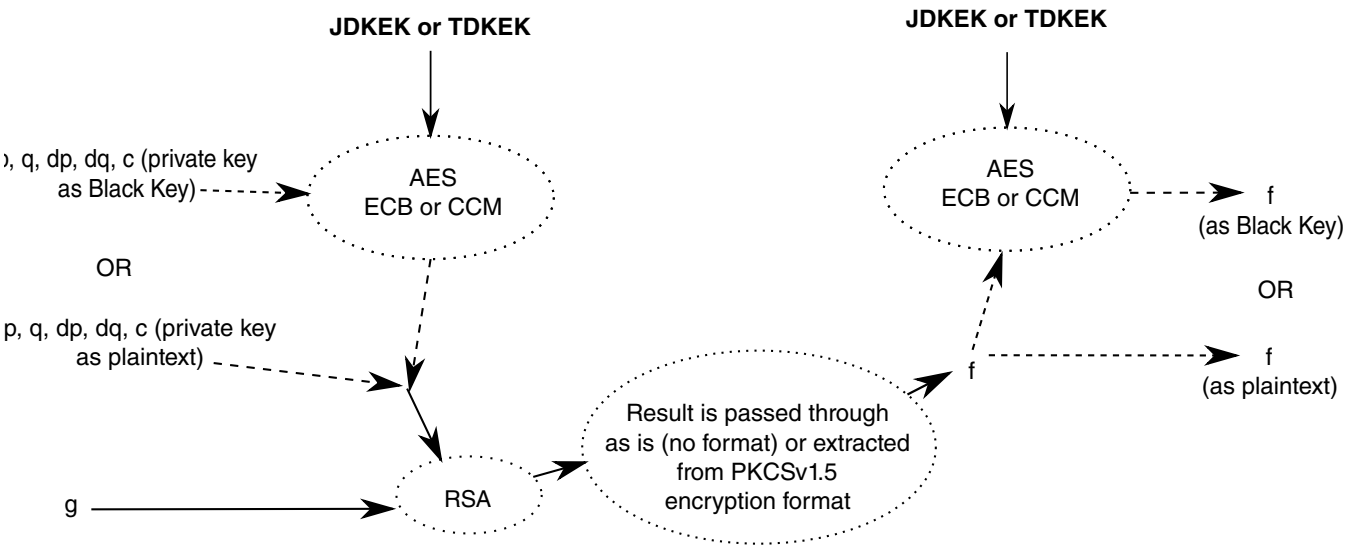


Figure 10-14. RSA Decrypt Operation - private key form #3

This figure shows the PDB for private key form #3. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 10-126. RSA decrypt PDB - private key form #3

SGF (9 bits)	Reserved (11 bits)	#n (12 bits)
Reference to g		
Reference to f		
Reference to c		
Reference to p		
Reference to q		
Reference to d_p		
Reference to d_q		
Reference to tmp*p*		
Reference to tmp*q*		
Reserved (8 bits)	#q (12 bits)	#p (12 bits)

The fields #n, #q and #p contain right-aligned, 12-bit values that indicate the size (plaintext sizes, in bytes) of n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. Note that #dp and #c are assumed to be #p, and #dq is assumed to be #q. tmp*p* needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp*q* needs to be as long as q (either #q, or, if q is encrypted, as big as the encrypted value of q).

Table 10-127. RSA Decrypt PDB - private key form #3; SGF Field

31	30	29	28	27	26	25	24	23
ref g	ref f	ref c	ref p	ref q	ref dp	ref dq	ref tmp*p*	ref tmp*q*
NOTE: If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.								

The RSA decrypt operation also accepts the private key in form #4, (p, q, dp, dq, cr, rrp, rrq).

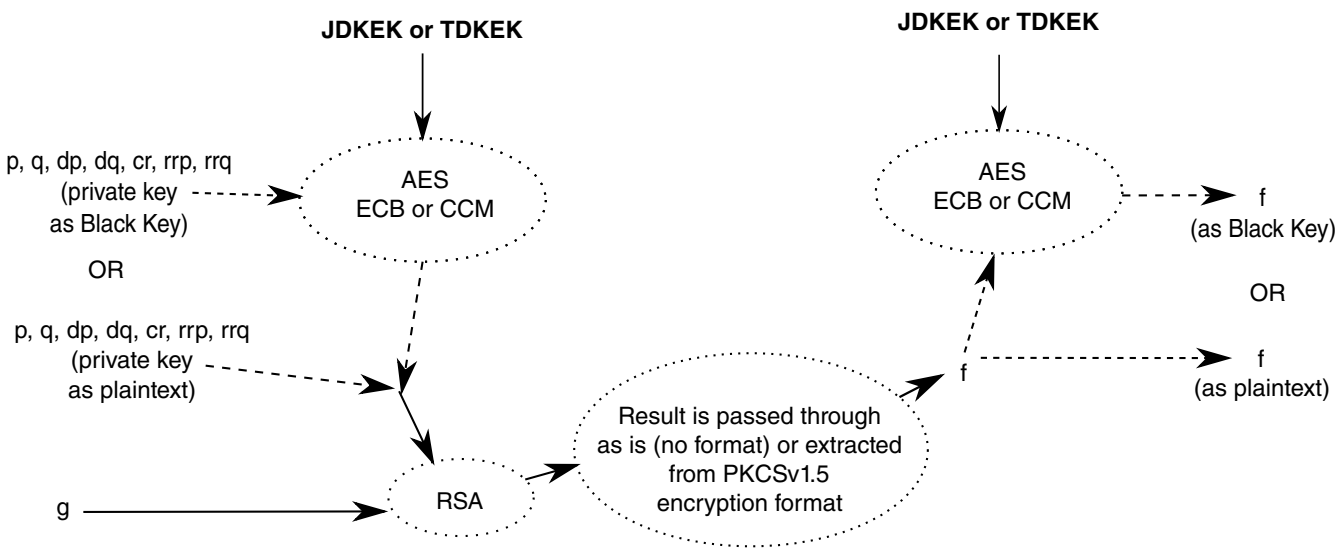


Figure 10-15. RSA Decrypt Operation - private key form #4

This figure shows the PDB for private key form #4. All references are address pointers.

Table 10-128. RSA decrypt PDB - private key form #4

SGF (10 bits)	Reserved (5 bits)	No_tm p*q* (1 bit)	Reserved (4 bits)	#n (12 bits)
Reference to g				
Reference to f				
Reference to cr				
Reference to p				
Reference to q				
Reference to d_p				
Reference to d_q				
Reference to rr_p				
Reference to rr_q				
Reference to tmp*q* (This word is omitted from the PDB if No_tmp*q*=1.)				

Table continues on the next page...

Table 10-128. RSA decrypt PDB - private key form #4 (continued)

Reserved (8 bits)	#q (12 bits)	#p (12 bits)

The fields #n, #q and #p contain right-aligned, 12-bit values that indicate the size (plaintext sizes, in bytes) of n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. Note that #dp, #cr, and # rr_p are assumed to be be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p), and #dq and #rr_q are assumed to be #q. tmp needs to be as long as q (either #q, or, if q is encrypted, as big as the encrypted value of q).

NOTE

tmp*q* may be omitted (i.e. no_tmp*q* set to 1) if #p and #q are each less than 2048. If no_tmp*q* = 1 and either #p or #q is greater than 256 an error will be generated.

Table 10-129. RSA Decrypt PDB - private key form #4; SGF Field

31	30	29	28	27	26	25	24	23	22
ref g	ref f	ref cr	ref p	ref q	ref dp	ref dq	ref rrp	ref rrq	ref tmp*q*
NOTE: If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.									

10.9 Key agreement functions

The CAAM protocol processing capabilities described in Protocol acceleration are centered on bulk data encryption and authentication. This section focuses on key pre-processing, which is another important part of protocol processing.

CAAM has the capability to pre-compute material derived from keys for HMAC and RC4 S-boxes. When sharing, using the derived material offers significant performance improvements.

10.9.1 Implementation of the derived key protocol

This protocol is available to assist with replacing a negotiated key with a derived form of that key. In particular, this protocol can be used for these tasks:

- Compute the Derived HMAC Key
- Compute an ARC4 S-Box from a key.

The use of the derived form of the key is mandatory for bulk-data protocols such as IPsec, where the use of the derived form provides a significant speed advantage.

The Derived Key Protocol (DKP) is designed to allow a negotiated key to be replaced with the derived form in-place in a shared descriptor. For example, an IPsec descriptor can be written to supply an immediate HMAC key in negotiated form as a parameter to the DKP operation command. The DKP computes the Derived HMAC Key, leaving the derived key in the Class 2 Key register, available for the subsequent IPsec command. Further, the DKP updates the descriptor, replacing the DKP operation command with the appropriate KEY command, and replacing the negotiated form of the key with the derived form of the key. It is the responsibility of the descriptor author to ensure the resulting derived key will not overwrite any descriptor commands that need to be kept.

10.9.1.1 Using DKP with HMAC keys

When used to generate HMAC keys, DKP receives an unprotected negotiated key and generates an unprotected Derived HMAC Key. DKP requires two block computations of the underlying hash function to generate the Derived HMAC Key, so the use of a Derived HMAC Key avoids those computations every time this key is used. This optimization is discussed further in Using the MDHA Key Register with Derived HMAC Keys. If an encrypted derived key is desired, or if an encrypted negotiated key is provided, see the FIFO STORE command and Output Data Types 16, 17, 26, 27 in Table 10-36.

When generating Derived HMAC Keys, the four-bit I/O control subfield of the PROTINFO field in the DKP Operation command is split in half; the upper 2 bits define the Input Source, and the lower two bits define the Output Destination. Not all combinations are valid.

Input Source - bits 16-17

Table 10-130. DKP input destination field

Setting	Description
00	IMM - negotiated key is in words immediately following the DKP Operation Command. This option can only be used with an Immediate Output Destination (OD=00).
01	SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command. This must be the choice when DKP is used in a trusted descriptor.
10	PTR - the input key is referenced by the address found immediately following the DKP Operation Command.
11	SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command.

Output Destination - bits 18-19

Table 10-131. DKP output destination field

Setting	Description
00	IMM - resulting Derived HMAC Key will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting Derived HMAC Key is twice the underlying hash context length. See Table 10-132 Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source.
01	SEQ - the resulting Derived HMAC Key will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source. This must be the choice when DKP is used in a trusted descriptor.
10	PTR - the resulting Derived HMAC Key will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF.
11	SGF - the resulting Derived HMAC Key will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR.

The twelve-bit length field designates the number of bytes the negotiated key takes. The length of the Derived HMAC Key is determined by the underlying hash function chosen, as shown.

Table 10-132. Lengths of the Derived HMAC Keys

Hashing algorithm	Length of Derived HMAC Key
MD5	32 bytes / 8 words
SHA-1	40 bytes / 10 words Note that while the utilized length of the derived HMAC SHA-1 key is 40 bytes, the key is stored in 48 bytes, with 4 bytes padding after each functional half.
SHA-224 SHA-256	64 bytes / 16 words
SHA-384 SHA-512	128 bytes / 32 words

10.9.1.2 Using DKP with ARC4 keys

When used to generate an ARC4 S-Box, DKP receives a negotiated key that may or may not have been encrypted, and uses AFHA to produce a derived key in the form of an S-Box.

The S-Box is too large to be made immediate. As a result, for ARC4 the I/O control field of the Operation Command has a different definition; some values will select encrypted keys.

Table 10-133. I/O control fields for ARC4

Bit Field	Description
16 KE	Keys Encrypted. Both the negotiated key and the derived key are encrypted. 0 - Neither key encrypted 1 - both negotiated and derived key are encrypted
17 EKT	Encrypted Key Type 0 - ECB-encrypted key 1 - CCM-encrypted key
18-19 SD	Source/Destination 00 - Simple pointers for source and destination -- address immediately after the DKP operation command is the memory location where the negotiated key is read and where the derived key is written. 01 - SGF pointers for source and destination -- address immediately after the DKP operation command is the memory location where a scatter/gather table resides. This scatter/gather table directs where the negotiated key is read and where the derived key is written. 10 - SEQ sequence for source and destination -- the negotiated key is read from the input frame as defined by the SEQ IN PTR command, and the derived key is written to the output frame as defined by the SEQ OUT PTR command. This must be the choice when DKP is used in a trusted descriptor. 11 - Reserved

10.9.1.3 Implementation of the Blob Protocol

The blob protocol provides a method for cryptographically protecting the confidentiality and integrity of user data across SoC power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the SoC is powered down. The key used to encrypt the blob is derived from a non-volatile master secret key so the blob can be decrypted when the SoC powers up again. More details on the Blob protocol can be found in section Blobs

10.10 Cryptographic hardware accelerators (CHAs)

This section describes the functionality of each individual CHA used by the DECO.

Table 10-134. Summary of cryptographic hardware accelerators (CHAs)

Definition	Abbreviation	What it implements	Cross-reference
Public-key hardware accelerator	PKHA	RSA, Diffie-Hellman, DSA, Elliptic-Curve Diffie-Hellman, Elliptic-Curve DSA	Public-key hardware accelerator (PKHA) functionality

Table continues on the next page...

Table 10-134. Summary of cryptographic hardware accelerators (CHAs) (continued)

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-224, SHA-256 authentication algorithms	Message digest hardware accelerator (MDHA) functionality
AES accelerator	AESA	The AES encryption algorithm	AES accelerator (AESA) functionality

10.10.1 Public-key hardware accelerator (PKHA) functionality

The PKHA module is capable of performing a number of different operations used in public-key cryptography, including modular arithmetic functions such as addition, subtraction, multiplication, exponentiation, reduction, squaring, cubing, simultaneous exponentiation, and inversion. All of these functions are provided in both integer and polynomial-binary field versions, except modular subtraction, which is the same as addition for binary polynomials. There are also elliptic-curve functions for point addition, point doubling, point validation, and point multiplication for the standard prime and binary curves.Most of these functions can be performed timing-equalized to thwart timing-related side-channel attacks. PKHA also includes a Miller-Rabin primality test function for detecting prime numbers.

The PKHA internally performs modular multiply operations using "Montgomery multiplication". For efficiency, many of these functions have a variant which allows either inputs or outputs in Montgomery form. Some have variants to supply the Montgomery conversion factor. These save time over the variations without. Internally, the PKHA operates on digits of these values. Different versions of the PKHA may have a different digit size. This PKHA has a digit size of 32 bits. This has implications for the inputs and outputs of certain functions. See the discussion on Montgomery arithmetic.

Because the numbers used in public-key cryptography are typically quite large and often referenced many times during a function, the inputs to PKHA are loaded into registers. PKHA has four of these registers labeled A, B, E, and N. A and B are for operands and results. E is for "keys", and N holds the modulus. For ECC functions, A and B are divided up into equal-size quadrants to accommodate the greater number of inputs required.

PKHA also has two other types of functions for manipulating the data in the registers. These are the Clear Memory and Copy Memory functions. The Clear Memory function allows all or any combination of the registers to be overwritten with zeros. The Copy Memory functions can be used to copy data from any of the A, B or N registers or register quadrant to any of the registers A, B, E or N.

PKHA requires that all data for a given function be loaded before the Mode Register is written to invoke a function. This convention indicates to PKHA that all needed data has been loaded, and the function can now be launched. The typical procedure for executing a PKHA function is to use KEY and FIFO LOAD Commands to load the registers (usually N first), followed by an PKHA OPERATION Command to write the Mode Register, followed by one or more FIFO STORE Commands to store the result. PKHA functions may also be cascaded, so that the output of one function stays in a PKHA memory to become an input for the next function.

When loading or storing a value, it is important that its associated size register not change during the operation. To help avoid this issue when loading ECC parameters, make sure that all quadrants of a given register have the same size values by left-filling "short" values with zero. If a size register for a FIFO LOAD command may change before it is complete, it is necessary to cause the Descriptor to stall until safe to proceed: insert a JUMP Command before the offending command: JUMP jsl = 1 type = 0 cond = nifp offset = 1 (instruction 0xA1000401). In the other case, where a FIFO STORE may still be in progress when a subsequent command will change the value in its size register, insert a SEQ FIFO STORE Command before the offending command: SEQ FIFO STORE length=0 (Instruction 0x68000000).

10.10.1.1 Modular math

Almost all math operations require with a modulus value in the N Memory. Math operations involving multiplication (multiplication, exponentiation, prime test, and ECC functions) are performed internally using Montgomery values.

10.10.1.2 About Montgomery values

The PKHA contains a Modular Arithmetic Unit. Multiplication is always modular multiplication:

$A * B \bmod N$.

The PKHA performs this computation with a Montgomery multiplier. A Montgomery multiplier can be more efficient than a multiply-then-reduce calculation because the modular reduction is done as part of the multiplication and the working product never gets larger than the modulus. In a normal multiplication, the product, before reduction, would be the size of the sum of the factors, so usually twice the size of the modulus. The factors in a Montgomery multiplication each have an R factor, and, as part of the multiplication and modular reduction, one R is removed. Thus, the computation performed is:

$(AR * BR) / R \bmod N$.

The equivalent of "division by R" occurs even if one of the inputs does not have an R factor.

A number of PKHA functions accept inputs in Montgomery form instead of normal values. Some instead take $R^2 \bmod N$ as an input. These functions can be faster than their normal-value alternatives if several operations are performed in a row or if these values are known in advance. This is because, before being used, ($R^2 \bmod N$ needs to be computed and) normal values need to be converted internally to Montgomery form.

The Montgomery form of a value is $value*R \bmod N$, referred to here as $\underline{value}$. The term $R = 2^{SD}$ is the Montgomery factor, where D is the digit size (of a digit in the PKHA arithmetic unit), in bits, and S is the minimum number of digits needed to hold the value in N. R is therefore dependent on N and D.

To use the PKHA to convert a normal value to a Montgomery value, one must first compute (or know) $R^2 \bmod N$, the Montgomery Conversion Factor. The following steps can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

R^2 = MOD_R2(N)

$\underline{A}$ = MOD_MUL_IM_OM(A, B=R^2, N)

The equivalent F2M function can be used for binary polynomial values.

Eventually, the value needs to be converted out of Montgomery form. This can be done by performing another multiply (R^2 is not needed for this).

```
A = MOD_MUL_IM_OM(A=A, B=1, N)
```

Another method is to cause the PKHA to perform a multiplication and conversion to normal form. Internally, there are two multiplications: first the two inputs, then the product by one.

```
AB mod N = MOD_MUL_IM(A=A, B=B, N)
```

A third method is to have just one factor (either one) in Montgomery form:

```
AB mod N = MOD_MUL_IM_OM(A=A, B=B, N)
```

The following operations can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

The equivalent F2M functions can be used for binary polynomial values.

It is possible to add and subtract Montgomery values, if *R* mod *N* is the same. Do not mix and match Montgomery and normal values for addition or subtraction. 5 + 3R => *5/R + 3R* or *5/R + 3*; neither is likely the desired result.

10.10.1.3 Non-modular Math

Although addition, subtraction, and multiplication functions require a modulus, it is possible to perform these calculations without any reduction: the modulus must be larger than the expected result.

For addition and subtraction, this is easily done. For multiplication, the MOD_MUL function may be used, but it is not the most efficient, as internally first R^2 mod N will be computed, then two multiplications will be performed (first to convert one factor into Montgomery, then to compute the product, not in Montgomery).

For non-modular multiplication, MUL_IM_OM is much more efficient, as only one multiplication will be performed. This can be used if the factors are not in Montgomery form, i.e., if the product to be calculated is *A*B* instead of *A*B mod N*. Since the multiplier always "divides by *R*', a special modulus value in Nram is required which will make *R* have the value 1. This is done by creating a modulus *N = R-1* so that *R mod N* will have the value one. This way, normal values are the same as Montgomery values; no conversion is necessary and the multiplier will quietly "divide by one" to no effect.

As an example, on a PKHA with a digit size of 32 bits and a product which will be no more than six bytes long, $R = 2^{SD} = 2^{2*32} = 2^{64}$. Therefore the modulus must be 0xFFFFFFFFFFFFFFFF.

For computation with binary polynomials, the equivalent F2M functions may be used.

10.10.1.4 Elliptic-Curve Math

The PKHA provides point math operations on different types of elliptic curves. These include the ability to add two points (+ operator), double a point, and multiply a point by an integer (scalar) value (x operator).

The input points are assumed to be valid points on the curve. If non-point coordinates or invalid curve parameters are used an input, then a non-point set of coordinates are likely to be returned as output. The "ECC Point Check" functions may be used to verify that a point's (x,y) values constitute a point which satisfies the equation for the curve.

The minimum modulus is 1 byte. The maximum modulus is 1024 bits, 128 in length, or one quadrant.

If xx is the Point at Infinity, *P*and *Q* are points on the curve, and *j* and *k* are integers, then the following identities, as well as others easily derived by taking advantage of associative and commutive properites, hold:

- $P + Q = Q + P$
- $P = xx + P$
- $xx = 0 \times P$
- $(j + k)P = (j \times P) + (k \times P)$

There may be times when the negative of a point is necessary:

- When subtracting points $P_A - P_B$
- When multiplying by a negative integer: $-abs(k) \times P_A$

To subtract, one can negate the second term and perform an addition, i.e.

$P_C = P_A - P_B = P_A + (-P_B)$

When multiplying by a negative value, one can either negate the starting point or the ending point. The multiplication value is the absolute value of the scalar, i.e., when *k* is negative

$P_C = k \times P_A = -abs(k) \times P_A = abs(k) \times (-1P_A) = -(abs(k) \times P_A)$

10.10.1.4.1 ECC_MOD: Point math on a standard curve over a prime field (F_p)

The ECC_MOD family of functions perform Add, Double, and scalar Multiply operations on points on a curve defined by the short Weierstrass equation:

$E: y^2 = x^3 + ax + b \bmod p$

where p is the a prime integer > 3. These operations are available in Affine Coordinates (*x,y*).

The modulus (value in N memory) for these operations is *p*, also referred to as *q*.

The equality for the negative of a point *P*, in affine coordinates, is *-P = -(x,y) = (x, -y)*

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, depends upon the type of curve and the value of the *b* term of the curve's equation:

- Where *b* is equal to 0: (0, 1)
- Where *b* is not equal to 0: (0, 0)

10.10.1.4.2 ECC_F2M: Point math on a standard curve over a binary field (F_{2^m})

The ECC_F2M family of functions perform add, double, and scalar multiply operations on points on a curve defined by the short Weierstrass equation:

E: $y^2 + xy = x^3 + ax^2 + b$

These operations are available in Affine Coordinates (*x,y*). All inputs and output values of polynomial values are in polynomial basis. For example, x^5+x+1 is represented as 23h

The modulus (value in N memory) for these functions is *q*, the field-defining irreducible polynomial for the curve. Other documents use other symbols, including *p(t)*, *f(t)*, and *f*.

The equality for the negative of a point, in affine coordinates, is *-P = -(x,y) = (x, x+y)*.

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

Because of the way the point operations are performed over a binary field, these functions require as an input the value *c* rather than *b*. The relationship between these two values is:

$b = c^4 \bmod q$

and

$c = b^{2^{m-2}} \bmod q$, where *m* is the degree (the power of its highest-power term) of *q*.

This *c* value is referred to as *b'* in the ECC Public-Key protocols for ECDSA sign, and so on. The calculation of *c* is expensive, so it is obviously an advantage to calculate it only once or have it precomputed. See Special values for common ECC domains for these values for common ECC domains.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, is: (0, 0).

10.10.1.5 PKHA Mode Register

The formats of the PKHA Mode Register are described in detail in PKHA OPERATION command.

The following tables list the valid PKHA_MODE values for all PKHA functions:

PKHA Clear Memory Functions: Table 10-137

PKHA Modular Arithmetic functions: Table 10-73

PKHA Elliptic Curve functions: PKHA OPERATION : Elliptic Curve Functions

PKHA Elliptic Curve functions: PKHA OPERATION : Elliptic Curve Functions

PKHA Copy Memory functions: Table 10-78

NOTE

Use of any PKHA_MODE value not listed in these tables results in an invalid mode error.

10.10.1.6 PKHA functions

The various PKHA functions are described in the following subsections. The following information applies to all PKHA functions.

- Mode Register bits that may be either 1's or 0's for the given function are represented with *x*.
- For convenience, in all the descriptions below the output is shown as the default B, although the actual output destination can be specified for most functions via the Class 1 Mode Register[OutSel] field to be either the B RAM or the A RAM.
- For each PKHA function, the specified mode bits are in the Class 1 Mode Register[PKHA_MODE_LS] field.

- For all of the PKHA functions, the Class 1 Mode Register[PKHA_MODE_MS] field is set to 8h.
- The descriptions specify the output register(s) and any other registers that might be modified. Note that the default output register is still modified but the output is placed into the specified destination register(s).
- Note that any parameter underlined is in Montgomery form (for example, $\underline{A}$ = AR mod N).
- Errors reported by PKHA are written to the Job Ring Output Status Register and termination status word (Job termination status/error codes). They are encoded in the ERRID field.
- Three flags in the CCB Status Register may be set by PKHA: PIZ, PIO, and PRM. These flags can be tested by the JUMP (HALT) command. is set to indicate that PKHA generated a result equal to zero, or, in the case of ECC functions, the point at infinity.
- PIO is set whenever a GCD routine finds that the Greatest Common Denominator of two numbers is the number 1. For other general non-ECC functions, it means that the result is equal to one. This may also be referred to as the GCD flag.
- PRM is set by the PRIME_TEST routine if it finds that a candidate integer is probably prime (that is, passes the Miller-Rabin primality test).
- It is important to note that the PKHA mathematical functions work in terms of "digits"; that is, the arithmetic unit is pipelined to work on a digit of data at a time. For PKHA-32 a digit = 32 bits (4 bytes) of data, for PKHA-64 a digit = 64 bits (8 bytes), and for PKHA-128 a digit = 128 bits (16 bytes). Therefore, the term 'digit' refers to 32, 64, or 128 bits of data in the input and/or output values used by the PKHA arithmetic unit.

10.10.1.6.1 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)

These functions copy data from a PKHA register (or register quadrant) specified as a source, to another PKHA register (or register quadrant) specified as a destination. COPY NSZ copies the amount of data specified by the N Size register. COPY_SSZ copies the amount of data specified in the source register's size register. The source and destination are specified in the Mode Value. The source can be A, B or N. The destination can be A, B, E or N, but not the same as the source.

In a quadrant copy, when NSZ/SSZ exceeds the length of a quadrant, the copy will carry on into the next (higher-numbered) quadrant(s).

When the copy operation has completed, the destination register's size register will be updated to contain the number of bytes copied.

Table 10-135. COPY_NSZ and COPY_SSZ function properties

<table>
<tr><th>Property</th><th>Notes</th></tr>
<tr><td>Mode value</td><td>
<table>
<tr><th>Bits 19:17</th><th>Bits 16,11,10</th><th>Bits 9:8</th><th>Bits 7:6</th><th>Bits 5:0</th></tr>
<tr><th>Source Register</th><th>Destination Register</th><th>Source Segment</th><th>Destination Segment</th><th>Function Code</th></tr>
<tr><td>000 = A Register</td><td>000 = A Register</td><td>00 = Segment 0</td><td>00 = Segment 0</td><td rowspan="4">01_0000 = Copy_NSZ

01_0001 = Copy_SSZ</td></tr>
<tr><td>001 = B Register</td><td>001 = B Register</td><td>01 = Segment 1</td><td>01 = Segment 1</td></tr>
<tr><td>011 = N Register</td><td>011 = N Register</td><td>10 = Segment 2</td><td>10 = Segment 2</td></tr>
<tr><td></td><td>010 = E Register</td><td>11 = Segment 3</td><td>11 = Segment 3</td></tr>
<tr><td>other values reserved</td><td>other values reserved</td><td></td><td></td><td></td></tr>
</table>
1. If the destination register is E, the source and destination seqments must be 00b.
</td></tr>
<tr><td>Input</td><td>None</td></tr>
<tr><td>Output</td><td>None</td></tr>
<tr><td>Requirements</td><td>For Copy_NSZ, the N-size Register must contain a valid value.
For Copy_SSZ, the source register's size register must contain a valid value.</td></tr>
<tr><td>Side effects</td><td>The destination register's size register is updated to the number of bytes copied.</td></tr>
<tr><td>Errors reported</td><td>None</td></tr>
<tr><td>Flags set</td><td>None</td></tr>
</table>

1. If the destination register is E, the source and destination seqments must be 00b.

10.10.1.6.2 Clear Memory (CLEAR_MEMORY) function

This function clears the specified registers or quadrants of registers in the PKHA. This includes the A, B, N and E. All registers or quadrants of registers are written with zeros.

A detailed description may be found in PKHA OPERATION : clear memory function.

Table 10-136. CLEAR_MEMORY function properties

Property	Notes
Mode value	*ABEN*_0000_00 *QQ_QQ* 00_0001, with the following restrictions on the combinations of ABEN and QQQQ: At least one of ABEN must be on. If E is on, all Q must be zero. Some example encodings are in the table below.
Input	None
Output	A = 0, B = 0, E = 0, N = 0, or some quadrant(s) thereof, as specified by ABEN and QQQQ. Each *Q* specifies a quadrant, in order from 3 through 0.
Requirements	The Mode Register specifies which registers or quadrants of registers to clear. If no quadrants are selected, then all quadrants of the specified register(s) are cleared.
Side effects	None
Errors reported	Invalid Mode, if no registers are selected, or E with one or more quadrants is selected
Flags set	None

Table 10-137. Example mode values for PKHA clear memory functions

Function name	Register selects				Quadrant selects				Brief description	Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex)
	A	B	E	N	3	2	1	0		
CLEAR_MEMORY	1	1	1	1	0	0	0	0	Clear registers A, B, E, N	F0001
	1	1	1	0	0	0	0	0	Clear registers A, B, E	E0001
	1	1	0	1	0	0	0	0	Clear registers A, B, N	D0001
	1	0	1	0	0	0	0	0	Clear registers A, E	A0001
	1	0	0	1	0	0	0	0	Clear registers A, N	90001
	0	1	0	1	0	0	0	0	Clear registers B, N	50001
	0	1	0	0	0	0	0	0	Clear register B	40001
	0	0	1	0	0	0	0	0	Clear register E	20001
	0	0	0	1	0	0	0	0	Clear register N	10001
	1	0	0	0	1	1	0	0	Clear quadrants 2 and 3 of register A	80301
	0	1	0	1	0	0	0	1	Clear quadrant 0 of registers B, N	50041
	1	1	0	0	1	0	0	0	Clear quadrant 3 of registers A, B	C0201

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

10.10.1.6.3 Arithmetic Functions

10.10.1.6.3.1 Integer Modular Addition (MOD_ADD) function

Table 10-138. MOD_ADD function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0010 (output placed in B) 0000_0000_000***1***_0000_0010 (output placed in A)
Input	• N = modulus and data size, any integer • A = first addend, any integer less than N • B = second addend, any integer less than N
Output	B (or A, if selected) = (A + B) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are each < N.
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.

Table continues on the next page...

Table 10-138. MOD_ADD function properties (continued)

Property	Notes
	If (A + B) >= N, N will be subtracted just once from the sum. That is, if (A + B) >= 2N, then the result will not be mod N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.2 Integer Modular Subtraction (MOD_SUB_1) function

Modular subtraction can be described as follows. If A ≥ B or A = B = 0, then B = A - B. Otherwise, if A < B, then B = A + N - B. The result is always positive and less than N.

Table 10-139. MOD_SUB_1 function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0011 (output placed in B) 0000_0000_000***1***_0000_0011 (output placed in A)
Input	• N = modulus, any integer • A = minuend, any integer less than N • B = subtrahend, any integer less than N
Output	B (or A, if selected) = (A - B) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are less than N.
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.3 Integer Modular Subtraction (MOD_SUB_2) function

Table 10-140. MOD_SUB_2 function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0100 (output placed in B) 0000_0000_000***1***_0000_0100 (output placed in A)
Input	• N = modulus, any integer • B = minuend, any integer less than or equal to N • A = subtrahend, any integer less than or equal to N
Output	B (or A, if selected) = (B - A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are < N

Table continues on the next page...

Table 10-140. MOD_SUB_2 function properties (continued)

Property	Notes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.4 Integer Modular Multiplication (MOD_MUL)

The (AB) mod N computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, MOD_MUL_IM or MOD_MUL_IM_OM) are more efficient. This function first computes R^2 mod N, then multiples one factor to produce AR, then multiplies AR*B to produce AB.

Table 10-141. MOD_MUL function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0101 (output placed in B) 0000_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, any odd integer • A = multiplicand, any integer less than N • B = multiplier, any integer less than N
Output	B (or A, if selected) = (AxB) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are < N.
Side effects	A and E are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N. • Divide-By-Zero Error is set if the most significant digit of the modulus is all zeros.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.5 Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM)

This function takes its inputs, integers, in Montgomery form, multiplies them modulo the value in the N register and returns the result as a field value. To do this, it performs two multiplications: AR*BR => ABR and ABR*1 => AB.

Table 10-142. MOD_MUL_IM function properties

Property	Notes
Mode value	1000_0000_000***0***_0000_0101 (output placed in B) 1000_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, any odd integer • A = multiplicand, a value in Montgomery form • B = multiplier, a value in Montgomery form
Output	B (or A if selected) = A x B mod N, the non-Montgomery product of the inputs
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are less than modulus N
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.6 Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function

This function performs the calculation A*B/R mod N, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is R2 mod N, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

Table 10-143. MOD_MUL_IM_OM function properties

Property	Notes
Mode value	1100_0000_000***0***_0000_0101 (output placed in B) 1100_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, any odd integer • A = multiplicand, a value in Mongomery format 0 ≤ A < N • B = multiplier, a value in Mongomery format 0 ≤ B < N
Output	B (or A, if selected) = (AxB) mod N

Table continues on the next page...

Table 10-143. MOD_MUL_IM_OM function properties (continued)

Property	Notes
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero.

10.10.1.6.3.7 Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ)

This function is commonly used to perform a single-step RSA operation. It computes R^2 mod N and converts A to Montgomery form before beginning the exponentiation. MOD_EXP_TEQ performs the same operation as MOD_EXP but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_TEQ, for a given modulus and size of exponent, is constant. In general MOD_EXP will run faster than MOD_EXP_TEQ, but will never run slower.

Table 10-144. MOD_EXP and MOD_EXP_TEQ function properties

Property	Notes
Mode value for MOD_EXP	0000_0000_0**00*0***_0000_0110 (output placed in B) 0000_0000_0**00*1***_0000_0110 (output placed in A)
Mode value for MOD_EXP_TEQ	0000_0000_0**10*0***_0000_0110 (output placed in B) 0000_0000_0**10*1***_0000_0110 (output placed in A)
Input	• N = modulus, any odd integer • A = an integer 0 ≤ A < N • E = exponent, any integer
Output	B (or A, if selected) = (A^E) mod N, a an integer 0 ≤ A < N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key (exponent) size = 512> bytes • A < N
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • Key Size Error is set if size of E = 0 or size of E > 512. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero.

10.10.1.6.3.8 Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function

This function is commonly used to perform a single-step RSA operation. It computes A = AR (mod N), where R is the Montgomery constant). The input data (base) to be exponentiated must be provided in the Montgomery form. The result will be returned in normal integer (non-Montgomery) representation. MOD_EXP_IM_TEQ performs the same operation as MOD_EXP_IM but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_IM_TEQ is constant for a given modulus and size of exponent. In general MOD_EXP_IM will run faster than MOD_EXP_IM_TEQ, but will never run slower.

Table 10-145. MOD_EXP_IM and MOD_EXP_IM_TEQ function properties

Property	Notes
Mode value for MOD_EXP_IM	1000_0000_0**00*0***_0000_0110 (output placed in B) 1000_0000_0**00*1***_0000_0110 (output placed in A)
Mode value for MOD_EXP_IM_TEQ	1000_0000_0**10*0***_0000_0110 (output placed in B) 1000_0000_0**10*1***_0000_0110 (output placed in A)
Input	• N = modulus, any odd integer • A = a value 0 ≤ A < N, in Montgomery form • E = exponent, any integer (normal integer representation)
Output	B (or A, if selected) = (A^E) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key (exponent) size = 512 bytes • A < N
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • Key Size Error is set if size of E = 0 or size of E > 512. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.9 Integer Simultaneous Modular Exponentiation (MOD_SML_EXP)

MOD_SML_EXP performs two modular exponentiations and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

Table 10-146. MOD_SML_EXP function properties

Property	Notes
Mode value	0000_0000_000***0***_0001_0110 (output placed in B) 0000_0000_000***1***_0001_0110 (output placed in A)

Table continues on the next page...

Table 10-146. MOD_SML_EXP function properties (continued)

Property	Notes
Input	• N = modulus, any odd integer • A0 = an integer < N, first base • E = an integer, first exponent • A2 = an integer < N, second base • B = an integer, second exponent
Output	B (or A if selected) = $A0^E$ * $A2^B$ mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 1/2 RAM size • Maximum key (exponent) size and B size = 1/4 RAM size • A0 < N • A2 < N • The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N
Side effects	A, B, and E are modified.
Errors reported	• N size error is set if N size > half RAM • A size error is set if A size > half RAM • B size error is set if B size > quarter-RAM. • E size error is set if E size > quarter-RAM
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.10 Integer Modular Square (MOD_SQR and MOD_SQR_TEQ)

This function may be used to square an integer value. MOD_SQR_TEQ, the timing equalized version, will take the same time to complete for a given modulus. In general the MOD_SQR version will run faster than MOD_SQR_TEQ, but will never run slower.

Table 10-147. MOD_SQR and MOD_SQR_EXP function properties

Property	Notes
Mode value for MOD_SQR	0000_0000_0**0**0***0***_0001_1010 (output placed in B) 0000_0000_0**0**0***1***_0001_1010 (output placed in A)
Mode value for MOD_SQR_TEQ	0000_0000_0**1**0***0***_0001_1010 (output placed in B) 0000_0000_0**1**0***1***_0001_1010 (output placed in A)
Input	• N = modulus, any odd integer • A = a field element
Output	B (or A, if selected) = (A*A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512.

Table continues on the next page...

Table 10-147. MOD_SQR and MOD_SQR_EXP function properties (continued)

Property	Notes
	• Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.11 Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_TEQ will take the same time to complete for any value of A. In general MOD_SQR_IM will run faster than MOD_SQR_IM_TEQ, and will never run slower.

Table 10-148. MOD_SQR_IM and MOD_SQR_IM_TEQ function properties

Property	Notes
Mode value for MOD_SQR_IM	1000_0000_0**00*0***_0001_1010 (output placed in B) 1000_0000_0**00*1***_0001_1010 (output placed in A)
Mode value for MOD_SQR_IM_TEQ	1000_0000_0**10*0***_0001_1010 (output placed in B) 1000_0000_0**10*1***_0001_1010 (output placed in A)
Input	• N = modulus, any odd integer • A = an integer < N in Montgomery form
Output	B (or A, if selected) = (A*A) mod N, an integer
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.12 Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_OM_TEQ will take the same time to complete for any value of A. In general MOD_SQR_IM_OM will run faster than MOD_SQR_IM_OM_TEQ, and will never run slower.

Table 10-149. MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ function properties

Property	Notes
Mode value for MOD_SQR_IM_OM	1100_0000_0**00*0***_0001_1010 (output placed in B) 1100_0000_0**00*1***_0001_1010 (output placed in A)
Mode value for MOD_SQR_IM_OM_TEQ	1100_0000_0**10*0***_0001_1010 (output placed in B) 1100_0000_0**10*1***_0001_1010 (output placed in A)
Input	• N = modulus, any odd integer • A = a Montgomery value < N
Output	B (or A, if selected) = (A*A) mod N, in Montgomery form
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = >Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.13 Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ)

This function may be used to cube an integer value. These functions first compute the Montgomery conversion factor, R^2 *mod N* and then convert the value to cube. For a given modulus, MOD_CUBE_TEQ will take the same time to complete for any value of A. In general MOD_CUBE will run faster than MOD_CUBE_TEQ, and will never run slower.

Table 10-150. MOD_CUBE and MOD_CUBE_TEQ function properties

Property	Notes
Mode value for MOD_CUBE	0000_0000_0**00*0***_0001_1011 (output placed in B) 0000_0000_0**00*1***_0001_1011 (output placed in A)
Mode value for MOD_CUBE_TEQ	0000_0000_0**10*0***_0001_1011 (output placed in B) 0000_0000_0**10*1***_0001_1011 (output placed in A)
Input	• N = modulus, any odd integer • A = a value < N
Output	B (or A, if selected) = (A*A*A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.

Table continues on the next page...

Table 10-150. MOD_CUBE and MOD_CUBE_TEQ function properties (continued)

Property	Notes
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.14 Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ)

MOD_CUBE_IM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM will run faster than MOD_CUBE_IM_TEQ, but will never run slower.

Table 10-151. MOD_CUBE_IM and MOD_CUBE_IM_TEQ function properties

Property	Notes
Mode value for MOD_CUBE_IM	1000_0000_0**00*0***_0001_1011 (output placed in B) 1000_0000_0**00*1***_0001_1011 (output placed in A)
Mode value for MOD_CUBE_IM_TEQ	1000_0000_0**10*0***_0001_1011 (output placed in B) 1000_0000_0**10*1***_0001_1011 (output placed in A)
Input	• N = modulus, any odd integer • A = a Montgomery value
Output	B (or A, if selected) = (A*A*A) mod N, a normal value
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.15 Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ)

MOD_CUBE_IM_OM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_OM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM_OM will run faster than MOD_CUBE_IM_OM_TEQ, but will never run slower.

Table 10-152. MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ function properties

Property	Notes
Mode value for MOD_CUBE_IM_OM	1100_0000_0**0**0***0***_0001_1011 (output placed in B) 1100_0000_0**0**0***1***_0001_1011 (output placed in A)
Mode value for MOD_CUBE_IM_OM_TEQ	1100_0000_0**1**0***0***_0001_1011 (output placed in B) 1100_0000_0**1**0***1***_0001_1011 (output placed in A)
Input	• N = modulus, any odd integer • A = a Montgomery value
Output	B (or A, if selected) = (A*A*A) mod N, a Montgomery value
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.16 Integer Modular Square Root (MOD_SQRT)

The modular square root function computes output result B, such that (B x B) mod N = input A. If no such B result exists, the result will be set to zero and the PKHA "prime" flag will be set. Otherwise, A is square, and the function will compute one of the two roots. The second square root (B') can be found by calculating B' = N - B. If A is zero, then there is one root, zero.

Input values N and A are limited to a maximum size of 128 bytes.

Table 10-153. MOD_SQRT function properties

Property	Notes
Mode value	0000_0000_000***0***_0001_0111 (output placed in B) 0000_0000_000***1***_0001_0111 (output placed in A)
Input	• N0 (N RAM, 1st quadrant) = modulus, an odd prime integer • A0 (A RAM, 1st quadrant) = input value, for which a square root is to be calculated
Output	B0 (or A0, if selected) is calculated such that (B0 x B0) mod N = A mod N. If no such B exists, then B is set to 0.
Requirements	• Minimum modulus size = 1 byte) • Maximum modulus size = 128 bytes • A is < N.
Side effects	All of the following PKHA RAM quadrants are modified: E2, E3, N1, N2, N3, B1, B2, B3, A1, A2, A3.

Table continues on the next page...

Table 10-153. MOD_SQRT function properties (continued)

Property	Notes
Errors reported	• Data Size Error is set if the size of N is greater than 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PRM is set if no square root solution can be found. ZERO is set if the result is zero (i.e., input A was zero).

10.10.1.6.3.17 Integer Modulo Reduction (MOD_AMODN)

MOD_AMODN computes the remainder of A divided by N. A and N can be of any size and it is not required that A > N, but N must be non-zero.

Table 10-154. MOD_AMODN function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0111 (output placed in B) 0000_0000_000***1***_0000_0111 (output placed in A)
Input	• N = modulus, any non-zero integer • A = any integer
Output	B (or A, if selected) = A mod N, A reduced modulo N
Requirements	• N = non-zero value • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Divide By Zero Error is set if N = 0.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.18 Integer Modular Inversion (MOD_INV)

MOD_INV computes the inverse of A, if an inverse exists. If the modulus, N, is prime, then all values of A, 1 ≤ A < N, are guaranteed to have an inverse mod N. If N is not prime, A may or may not have an inverse. It will have one only if GCD(A, N) == 1.

Table 10-155. MOD_INV function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1000 (output placed in B) 0000_0000_000***1***_0000_1000 (output placed in A)
Input	• N = modulus, any non-zero integer • A = any non-zero integer lass than N

Table continues on the next page...

Table 10-155. MOD_INV function properties (continued)

Property	Notes
Output	B (or A, if selected) = A^{-1} mod N, an integer, the multiplicative inverse of A
Requirements	• Neither A or N can be zero. • A must be less than N. • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	A and E are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0, or if there is no inverse.
Flags set	None

10.10.1.6.3.19 Integer Montgomery Factor Computation (MOD_R2)

This function is used to compute a constant to assist in converting operands into the Montgomery residue system representation. The constant R^2(mod N) is dependent upon the digit size of the PKHA and the value in N.

MUL, EXP, and ECC functions that do not have "IM" (Montgomery inputs) or an R2 input will internally invoke this routine to determine the constant and do the conversions before other operations.

If the modulus N is a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a not-insignificant amount of time to complete.

Table 10-156. MOD_R2 function properties

Property	Notes
Mode value	1000_0000_000***0***_0000_1100 (output placed in B) 1000_0000_000***1***_0000_1100 (output placed in A)
Input	N = modulus, any odd integer
Output	B (or A, if selected) = R2 mod N, where R = 2^{SD} where S is size of a digit in bits and D is the number of digits of N; in other words, D = ceiling [sizeof(N) in bits / S]
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even.
Flags set	None

10.10.1.6.3.20 Integer $R_E R_P$ mod P (MOD_RR)

This function is used to compute a constant to assist in converting operands into the Montgomery residue system representation specifically for Chinese Remainder Theorem while performing RSA with a CRT implementation where a modulus E=P x Q, and P and Q are prime numbers. Although labeled $R_E R_P$ mod P, this routine (function) can also compute $R_E R_Q$ mod Q.

To use this routine, the size of the prime number (either P or Q) in bytes must be written into the Data Size Register, and the size of the modulus E = P x Q in bytes must be written into the Key Size Register. Next, the prime modulus (either P or Q) is written into N-memory.

Table 10-157. MOD_RR function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1101 (output placed in B) 0000_0000_000***1***_0000_1101 (output placed in A)
Input	• N = modulus P or Q of CRT, an odd integer • Key size = number of bytes of E = P x Q (this size must be given, though content of E itself is not used)
Output	B (or A, if selected) = $R_E R_P$ mod P, where $R_E = 2^{SD(E)}$, and $R_P = 2^{SD(P)}$, where S is the size of a digit (in bits). D(E), D(P) are the number of digits of E, P respectively; in other words, D(E or P) = ceiling [size of (E or P) in bits / S]
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key size (P-size + Q-size) = 512 bytes • Key size > modulus size
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • Key Size Error is set if size of E = 0 or size of E > 512.
Flags set	None

10.10.1.6.3.21 Integer Greatest Common Divisor (MOD_GCD)

MOD_GCD finds the greatest common divisor of two integers.

Table 10-158. MOD_GCD function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1110 (output placed in B) 0000_0000_000***1***_0000_1110 (output placed in A)
Input	• N = any integer. The most-significant digit of N must be non-zero. • A = any integer less than or equal to N
Output	B (or A, if selected) = GCD(A,N), an integer less than or equal to A that divides both A and N

Table continues on the next page...

Table 10-158. MOD_GCD function properties (continued)

Property	Notes
	If the output is placed in B, the MOD_INV result is available in A.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and N may not both be zero
Side effects	A is modified
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0.
Flags set	PIO is set if the result is 1.

10.10.1.6.3.22 Miller_Rabin Primality Test (PRIME_TEST)

Table 10-159. PRIME_TEST function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1111 (output placed in B) 0000_0000_000***1***_0000_1111 (output placed in A)
Input	• N[1] = Candidate prime integer • A = An initial random seed for the base value of exponentiation; can be any integer 2 < A < N - 2 • B = "t" parameter, which is the number of trial runs. By default, it is set at 1 or B[7:0], whichever is greater. Only the lowest byte of the supplied value is used.
Output	B (or A, if selected) = 1 if candidate is believed to be prime, otherwise 0
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and N may not both be zero
Side effects	N and A are modified
Errors reported	• N Size Error is set if size of N = 0 or size of N > 512. • B Size Error is set if N size > 256. and the least-significant byte of B > 31. • Divide-By-Zero Error is set if no seed can be found that is in the legal range of 2 < A < N-2. This occurs if N = 1 or N = 3.
Flags set	PRM is set if the candidate is believed to be prime

1. If the most significant digit of N is zero, the result is always composite, the output is the value zero, and the PRM flag is not set, regardless of the primality of the value of N.

10.10.1.6.3.23 Right Shift A (RIGHT_SHIFT_A) function

Table 10-160. RIGHT_SHIFT_A function properties

Property	Notes
Mode value	0000_0000_000***0***_0001_1101 (output placed in B) 0000_0000_000***1***_0001_1101 (output placed in A)
Input	• A = first addend, any integer less than N
Output	B (or A, if selected) = (A + B) mod N

Table continues on the next page...

Table 10-160. RIGHT_SHIFT_A function properties (continued)

Property	Notes
Requirements	• Minimum A size = 1 byte • Maximum A size = 512 bytes
Side effects	None
Errors reported	• A Size Error is set if the size of A is greater than max size. If (A + B) >= N, N will be subtracted just once from the sum. That is, if (A + B) >= 2N, then the result will not be mod N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.24 Compare A B (COMPARE) function

Table 10-161. COMPARE function properties

Property	Notes
Mode value	0000_0000_0000_0001_1110
Input	• A = Value to be compared • B = Value to be compared • A Size = the number of least-significant bits that will be compared
Output	None (other than flags)
Requirements	A Size must be >= B Size.
Side effects	none
Errors reported	If B Size > A Size a "B Size" error will be generated.
Flags set	• PKHA_GCD_ONE is set if B > A • PKHA_ZERO is set if B == A • no flag is set if B < A

10.10.1.6.3.25 Evaluate A (EVALUATE) function

Table 10-162. EVALUATE function properties

Property	Notes
Mode value	***SB***00_0000_000***0***_0001_1111 (output placed in B) ***SB***00_0000_000***1***_0001_1111 (do not modify B) • If the ***S*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0**sss**, where **sss** is the updated A Size. • If the ***B*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_**bbbb**, where **bbbb** is the updated number of bits in A. • If the ***S*** bit is set and the ***B*** bit is set, the A-Size DWord will be pushed before the number-of-bits-in-A Dword.
Input	• A = Value to be evaluated
Output	• A Size is updated with the number of least-significant non-zero bytes, i.e. the position of the most-significant non-zero byte (least-significant byte is byte-position 0). This evaluation considers only the bytes specified by the incoming value of A Size. This allows the incoming value of A Size to be

Table continues on the next page...

Table 10-162. EVALUATE function properties (continued)

Property	Notes
	set so that A0 will be evaluated, ignoring the values in A1, A2 and A3, or the incoming value could be set so that (A1,A0) will be evaluated, ignoring A2 and A3, or (A2,A1,A0) could be evaluated, ignoring A3. • If the mode value specifies that the output is to be placed in B, the updated value of A Size will be copied into the least-significant two bytes of B and B Size will be set to 2. If the "do not modify B" option is selected, the updated value of A Size will not be copied into B.
Requirements	none
Side effects	• A Size will be modified. • B Size may be modified. • One or two DWords mmay be pushed to the output FIFO.
Errors reported	none
Flags set	• PKHA_GCD_ONE is set if A == 1 • PKHA_ZERO is set if A == 0. A Size (and number of bits in A) will be set to 0. Note that this could cause an A Size error in a subsequent PKHA operation.

10.10.1.6.3.26 Binary Polynomial (F_{2^m}) Addition (F2M_ADD) function

This function performs binary polynomial modular addition without any modulo reduction, as the value in the N register is ignored. Only its size is used, to determine the size of the result.

This type of addition is the equivalent of a bitwise XOR and this function may be used for that purpose.

This function could as easily be labeled F2M_SUB, as it is mathematically equivalent.

Table 10-163. F2M_ADD function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_0010 (output placed in B) 0000_0000_000***1***_0000_0010 (output placed in A)
Input	• Size of N (modulus size) • A = first addend, a binary polynomial • B = second addend, a binary polynomial
Output	B (or A, if selected) = A xor B, a binary polynomial
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • The N need not be written, because its contents are ignored, but the size of N must be written. This size is needed because inputs A and B are considered binary polynomials modulo some irreducible polynomial N.
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.27 Binary Polynomial (F_{2^m}) Modular Multiplication (F2M_MUL)

The (AB) mod N computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, F2M_MUL_IM or F2M_MUL_IM_OM) are more efficient. This function first computes R^2 mod N, then multiples $A*R^2$ to produce AR, then multiplies AR*B to produce AB.

Table 10-164. F2M_MUL function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_0101 (output placed in B) 0010_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, an irreducible polynomial • A = multiplicand, a field element • B = multiplier, a field element
Output	B (or A, if selected) = (AB) mod N, a field element
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are field elements modulo N.
Side effects	A and E are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.28 Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function

This function takes its inputs, binary polynomials, in Montgomery form, multiplies them modulo the value in the N register, used as a reduction polynomial, and returns the result as a field value. To do this, it performs two multiplications: AR*BR => ABR and ABR*1 => AB.

Table 10-165. F2M_MUL_IM function properties

Property	Notes
Mode value	1010_0000_000***0***_0000_0101 (output placed in B) 1010_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = multiplicand, a binary polynomial < 2^m, in Montgomery form • B = multiplicand, a binary polynomial < 2^m, in Montgomery form

Table continues on the next page...

Table 10-165. F2M_MUL_IM function properties (continued)

Property	Notes
Output	B (or A, if selected) = (AxB) mod N, a a binary polynomial < 2^m, non-Montgomery form
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are field elements in Montgomery form and must be modulo reduced by irreducible polynomial N.
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.29 Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function

This function performs the calculation A*B/R mod N, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is R2 mod N, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

Table 10-166. F2M_MUL_IM_OM function properties

Property	Notes
Mode value	1110_0000_000***0***_0000_0101 (output placed in B) 1110_0000_000***1***_0000_0101 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial 0 < 2^m, in Montgomery form. • B = a binary polynomial 0 < 2^m, in Montgomery form.
Output	B (or A, if selected) = (AxB) mod N, in Montgomery form
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are field elements in Montgomery form.
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even.

Table continues on the next page...

Table 10-166. F2M_MUL_IM_OM function properties (continued)

Property	Notes
	• A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.30 Binary Polynomial (F_{2^m}) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)

This function is similar to MOD_EXP but works on binary polynomials. It is provided mainly to assist in the computation of elliptic curve parameter "c", where c = $b^{2^{m-2}}$ mod n) given an elliptic curve parameter "b" and the field-defining polynomial in N. It computes R^2 mod N and converts A to Montgomery form before beginning the exponentiation. F2M_EXP_TEQ performs the same operation as F2M_EXP but with an added timing equalization security feature. Its exponentiation run-time, for a given modulus and size of exponent, is constant. In general F2M_EXP will run faster than F2M_EXP_TEQ, but will never run slower.

Table 10-167. F2M_EXP function properties

Property	Notes
Mode value for F2M_EXP	0010_0000_0**0**0***0***_0000_0110 (output placed in B) 0010_0000_0**0**0***1***_0000_0110 (output placed in A)
Mode value for F2M_EXP_TEQ	0010_0000_0**1**0***0***_0000_0110 (output placed in B) 0010_0000_0**1**0***1***_0000_0110 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial < 2^m • E = exponent, any integer
Output	B (or A, if selected) = (A^E) mod N, a binary polynomial
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key (exponent) size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • Key Size Error is set if size of E = 0 or size of E > 512. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.31 Binary Polynomial (F_{2^m}) Simultaneous Modular Exponentiation (F2M_SML_EXP)

F2M_SML_EXP performs two modular exponentiations on binary polynomials, and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

Table 10-168. F2M_SML_EXP function properties

Property	Notes
Mode value	0000_0000_000***0***_0001_0110 (output placed in B) 0000_0000_000***1***_0001_0110 (output placed in A)
Input	• N = modulus, any odd integer • A0 = a field element, first base • E = an integer, first exponent • A2 = a field element, second base • B = an integer, second exponent
Output	B (or A if selected) = $A0^E * A2^B$ mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 1/2 RAM • Maximum key (exponent) size and B size = 1/4 RAM size • A0 < N • A2 < N • The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N
Side effects	A, B, and E are modified.
Errors reported	• N size error is set if N size > half RAM • A size error is set if A size > half RAM • B size error is set if B size > quarter-RAM • E size error is set if E size > quarter-RAM
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.32 Binary Polynomial (F_{2^m}) Modular Square (F2M_SQR and F2M_SQR_TEQ)

F2M_SQR may be used to square an binary polynomial value. The timing equalized version, F2M_SQR_TEQ, will also square a binary polynomial value but for a given modulus will always take the same time to complete. F2M_SQR will usually run faster than F2M_SQR_TEQ, but will never run slower

Table 10-169. F2M_SQR and F2M_SQR_TEQ function properties

Property	Notes
Mode value for F2M_SQR	0010_0000_0**0**0***0***_0001_1010 (output placed in B) 0010_0000_0**0**0***1***_0001_1010 (output placed in A)

Table continues on the next page...

Table 10-169. F2M_SQR and F2M_SQR_TEQ function properties (continued)

Property	Notes
Mode value for F2M_SQR_TEQ	0010_0000_010***0***_0001_1010 (output placed in B) 0010_0000_010***1***_0001_1010 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m
Output	B (or A, if selected) = (A*A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.33 Binary Polynomial (F_{2^m}) Modular Square, Montgomery Input (F2M_SQR_IM and F2M_SQR_IM_TEQ)

F2M_SQR_IM may be used to square a binary polynomial in Montgomery form. F2M_SQR_IM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM will generally run faster than F2M_SQR_IM_TEQ, but will never run slower.

Table 10-170. F2M_SQR_IM and F2M_SQR_IM_TEQ function properties

Property	Notes
Mode value for F2M_SQR_IM	1010_0000_000***0***_0001_1010 (output placed in B) 1010_0000_000***1***_0001_1010 (output placed in A)
Mode value for F2M_SQR_IM_TEQ	1010_0000_010***0***_0001_1010 (output placed in B) 1010_0000_010***1***_0001_1010 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m in Montgomery form
Output	B (or A, if selected) = (A*A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero.

Table 10-170. F2M_SQR_IM and F2M_SQR_IM_TEQ function properties

Property	Notes
	GCD is set if the result is one.

10.10.1.6.3.34 Binary Polynomial (F_{2^m}) Modular Square, Montgomery Input and Output (F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ)

F2M_SQR_IM_OM may be used to square a binary polynomial in Montgomery form, and will output the result in Montgomery form. F2M_SQR_IM_OM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM_OM will generally run faster than F2M_SQR_IM_OM_TEQ, but will never run slower.

Table 10-171. F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ function properties

Property	Notes
Mode value for F2M_SQR_IM_OM	1110_0000_0**00*0***_0001_1010 (output placed in B) 1110_0000_0**00*1***_0001_1010 (output placed in A)
Mode value for F2M_SQR_IM_OM _TEQ	1110_0000_0**10*0***_0001_1010 (output placed in B) 1110_0000_0**10*1***_0001_1010 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m, possibly in Montgomery form
Output	B (or A, if selected) = (A*A) mod N, in Montgomery form
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.35 Binary Polynomial (F_{2^m}) Modular Cube (F2M_CUBE and F2M_CUBE_TEQ)

F2M_CUBE may be used to cube a binary polynomial value. The function will first compute the Montgomery conversion factor, $R^2 \ mod \ N$ and convert the value to cube. F2M_CUBE_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE will generally run faster than F2M_CUBE_TEQ, but will never run slower.

Table 10-172. F2M_CUBE and F2M_CUBE_TEQ function properties

Property	Notes
Mode value for F2M_CUBE	0010_0000_0**00*0***_0001_1011 (output placed in B) 0010_0000_0**00*1***_0001_1011 (output placed in A)
Mode value for F2M_CUBE_TEQ	0010_0000_0**10*0***_0001_1011 (output placed in B) 0010_0000_0**10*1***_0001_1011 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order *m* • A = a binary polynomial of order *<m*
Output	B (or A, if selected) = (A*A*A) mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.36 Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input (F2M_CUBE_IM and F2M_CUBE_IM_TEQ)

F2M_CUBE_IM may be used to cube a binary polynomial value that is in Montgomery form. F2M_CUBE_IM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM will generally run faster than F2M_CUBE_IM_TEQ, but will never run slower.

Table 10-173. F2M_CUBE_IM and F2M_CUBE_IM_TEQ function properties

Property	Notes
Mode value for F2M_CUBE_IM	1010_0000_0**00*0***_0001_1011 (output placed in B) 1010_0000_0**00*1***_0001_1011 (output placed in A)
Mode value for F2M_CUBE_IM_TEQ	1010_0000_0**10*0***_0001_1011 (output placed in B) 1010_0000_0**10*1***_0001_1011 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m in Montgomery form
Output	B (or A, if selected) = (A*A*A) mod N, a binary polynomial
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.

Table continues on the next page...

Table 10-173. F2M_CUBE_IM and F2M_CUBE_IM_TEQ function properties (continued)

Property	Notes
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.37 Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ)

F2M_CUBE_IM_OM may be used to cube a binary polynomial value that is in Montgomery form, and output the result in Montgomery form. F2M_CUBE_IM_OM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM_OM will generally run faster than F2M_CUBE_IM_OM_TEQ, but will never run slower.

Table 10-174. F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_EXP function properties

Property	Notes
Mode value for F2M_CUBE_IM_OM	1110_0000_0**00*0***_0001_1011 (output placed in B) 1110_0000_0**00*1***_0001_1011 (output placed in A)
Mode value for F2M_CUBE_IM_OM_TEQ	1110_0000_0**10*0***_0001_1011 (output placed in B) 1110_0000_0**10*1***_0001_1011 (output placed in A)
Input	• N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m, possibly in Montgomery form
Output	B (or A, if selected) = (A*A*A) mod N, in Montgomery form
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

10.10.1.6.3.38 Binary Polynomial (F_{2^m}) Modulo Reduction (F2M_AMODN)

F2M_AMODN computes the remainder of A divided by N. This is the equivalent of the MOD_AMODN function applied to a binary polynomial. A and N can be of any size and it is not required that A > N, but N must be non-zero.

Table 10-175. F2M_AMODN function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_0111 (output placed in B) 0010_0000_000***1***_0000_0111 (output placed in A)
Input	• N = modulus, any non-zero polynomial • A = any polynomial
Output	B (or A, if selected) = A mod N, a polynomial, binary element modulo N
Requirements	• N = non-zero value • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Divide By Zero Error is set if N = 0.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

10.10.1.6.3.39 Binary Polynomial (F_{2^m}) Modular Inversion (F2M_INV)

F2M_INV computes the multiplicative inverse of a binary polynomial.

Table 10-176. F2M_INV function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_1000 (output placed in B) 0010_0000_000***1***_0000_1000 (output placed in A)
Input	• N = modulus, an irreducible polynomial • A = a field element
Output	B (or A, if selected) = A^{-1} mod N, a field element, the multiplicative inverse of A
Requirements	• A is an element of the binary polynomial field. • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	A and E are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0.
Flags set	None

10.10.1.6.3.40 Binary Polynomial (F_{2^m}) R^2 Mod N (F2M_R2) Function

This function is used to compute the Montgomery Conversion Factor, which is used to convert operands into the Montgomery residue system representation. The constant R^2(mod N) is dependent upon the digit size of the PKHA and the value of N. If this value is not available, then this routine (function) is called to determine the constant before

other operations. If N contains a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a considerable amount of time to complete.

Table 10-177. F2M_R2 function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_1100 (output placed in B) 0010_0000_000***1***_0000_1100 (output placed in A)
Input	N = modulus, an irreducible polynomial
Output	B (or A, if selected) = R2 mod N, where R = 2^{SD} where S is size of a digit in bits and D is the number of digits of an irreducible polynomial, in other words D = ceiling [sizeof(N) in bits / S]
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even.
Flags set	None

10.10.1.6.3.41 Binary Polynomial (F_{2^m}) Greatest Common Divisor (F2M_GCD) Function

MOD_GCD finds the greatest common divisor of two binary polynomials.

Table 10-178. F2M_GCD function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_1110 (output placed in B) 0010_0000_000***1***_0000_1110 (output placed in A)
Input	• N = any polynomial. The most-significant digit of N must be non-zero. • A = any polynomial with degree less than or equal to N
Output	B (or A, if selected) = BINARY_GCD(A,N), a polynomial with degree less than or equal to polynomial A that divides both A and N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and N may not both be zero
Side effects	A is modified
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0.
Flags set	PIO is set if the result is 1.

10.10.1.6.4 Elliptic Curve Functions

10.10.1.6.4.1 ECC F_p Point Add, Affine Coordinates (ECC_MOD_ADD) Function

ECC_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 10-179. ECC_MOD_ADD function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1001 (output placed in B) 0000_0000_000***1***_0000_1001 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

10.10.1.6.4.2 ECC F_p Point Add, Affine Coordinates, R^2 Mod N Input (ECC_MOD_ADD_R2) Function

ECC_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECC_MOD_ADD_R2 has R^2 mod N as an additional input, this function is more efficient than ECC_MOD_ADD, which first must compute R^2 mod N before performing the addition.

Table 10-180. ECC_MOD_ADD_R2 function properties

Property	Notes
Mode value	0001_0000_000***0***_0000_1001 (output placed in B) 0001_0000_000***1***_0000_1001 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero

Table continues on the next page...

Table 10-180. ECC_MOD_ADD_R2 function properties (continued)

Property	Notes
	• [A0, A1] = first addend point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend point in affine coordinates (x,y) • B3 = R2 (R^2 mod N) input
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

10.10.1.6.4.3 ECC F_p Point Double, Affine Coordinates (ECC_MOD_DBL) Function

ECC_MOD_DBL computes the double (B + B) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 10-181. ECC_MOD_DBL function properties

Property	Notes
Mode value	0000_0000_000***0***_0000_1010 (output placed in B) 0000_0000_000***1***_0000_1010 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1, A2] = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = input point in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [B1, B2] + [B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates (x, y).
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even.

Table continues on the next page...

Table 10-181. ECC_MOD_DBL function properties (continued)

Property	Notes
	• A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

10.10.1.6.4.4 ECC F_p Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function

ECC_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECC_MOD_MUL in general will run faster than ECC_MOD_MUL_TEQ, but will never run slower.

Table 10-182. ECC_MOD_MUL and ECC_MOD_MUL_TEQ function properties

Property	Notes
Mode value for ECC_MOD_MUL	0000_0000_0**0**0***0***_0000_1011 (output placed in B) 0000_0000_0**0**0***1***_0000_1011 (output placed in A)
Mode value for ECC_MOD_MUL_TEQ	0000_0000_0**1**0***0***_0000_1011 (output placed in B) 0000_0000_0**1**0***1***_0000_1011 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero • E = scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0).
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(-k). After the computation is complete, the formula -P= (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

10.10.1.6.4.5 ECC F_p Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function

ECC_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECC_MOD_MUL_R2 has R^2 mod N as an additional input, this function is more efficient than ECC_MOD_MUL, which first must compute R^2 mod N before performing the multiplication. ECC_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECC_MOD_MUL_R2 in general will run faster than ECC_MOD_MUL_R2_TEQ, but will never run slower.

Table 10-183. ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECC_MOD_MUL_R2	0001_0000_0**0**0***0***_0000_1011 (output placed in B) 0001_0000_0**0**0***1***_0000_1011 (output placed in A)
Mode value for ECC_MOD_MUL_R2_TEQ	0001_0000_0**1**0***0***_0000_1011 (output placed in B) 0001_0000_0**1**0***1***_0000_1011 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = R2 mod N, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N.

Table continues on the next page...

Table 10-183. ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ function properties (continued)

Property	Notes
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

10.10.1.6.4.6 ECC F_p Check Point (ECC_MOD_CHECK_POINT) Function

ECC_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^2 = x^3 + ax + b$.

ECC_MOD_CHECK_POINT_R2 has R^2 mod N as an additional input, so it is more efficient than this function, which first must compute R^2 mod N before performing the operation.

Table 10-184. ECC_MOD_CHECK_POINT function properties

Property	Notes
Mode value	0000_0000_0000_0001_1100
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	• B1 = R^2 mod N, as described in MOD_R2MODN (Note that if the input point is invalid B1 may not be generated. Software should check the range and validity of x and y for the curve's equation before running this function.)

Table continues on the next page...

Table 10-184. ECC_MOD_CHECK_POINT function properties (continued)

Property	Notes
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

ECC_MOD_CHECK_POINT checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 = x^3 + ax + b \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

10.10.1.6.4.7 ECC F_p Check Point, R^2 Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function

ECC_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^2 = x^3 + ax + b$. ECC_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 \bmod N$ and $x^3 + ax + b \bmod N$. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. Since this function takes $R^2 \bmod N$ as an additional input, ECC_MOD_CHECK_POINT_R2 is more efficient than ECC_MOD_CHECK_POINT, which first must compute $R^2 \bmod N$ before performing the point check.

Table 10-185. ECC_MOD_CHECK_POINT_R2 function properties

Property	Notes
Mode value	0001_0000_0000_0001_1100
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = $R^2 \bmod N$, pre-computed as described in MOD_R2MODN (0Eh) • B2 = ignored

Table continues on the next page...

Table 10-185. ECC_MOD_CHECK_POINT_R2 function properties (continued)

Property	Notes
	• B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. (Note that if the input point is invalid this function may fail to detect "out of range" values that will still satisfy the curve equation. Software should check the range and validity of x and y for the curve's equation before running this function.) • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

ECC_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 = x^3 + ax + b \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

10.10.1.6.4.8 ECC F_{2^m} Point Add, Affine Coordinates (ECC_F2M_ADD) Function

ECC_F2M_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 10-186. ECC_F2M_ADD function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_1001 (output placed in B) 0010_0000_000***1***_0000_1001 (output placed in A)
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter, where c = $b^{2^{m-2}} \bmod n$ • B1, B2] = second addend in affine coordinates • B3 = ignored
Output	P[B1, B2] (or P[A0, A1], if A output selected) = P[A0, A1] + P[B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte

Table continues on the next page...

Table 10-186. ECC_F2M_ADD function properties (continued)

Property	Notes
	• Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1, and B2 and elliptic curve parameters A3 and B0 are elements of the binary polynomial field N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide By Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	None

10.10.1.6.4.9 ECC F_{2^m} Point Add, Affine Coordinates, R^2 Mod N Input (ECC_F2M_ADD_R2) Function

ECC_F2M_ADD_R2 performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates. Since this function takes R^2 mod N as an additional input, ECC_F2M_ADD_R2 is more efficient than ECC_F2M_ADD, which first must compute R^2 mod N before performing the addition.

Table 10-187. ECC_F2M_ADD_R2 function properties

Property	Notes
Mode value	0011_0000_000***0***_0000_1001 (output placed in B) 0011_0000_000***1***_0000_1001 (output placed in A)
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter, where c = $b^{2^{m-2}}$ mod n. Must not be zero. • [B1, B2] = second addend in affine coordinates • B3 = R2 input
Output	P[B1, B2] (or P[A0, A1], if A output selected) = P[A0, A1] + P[B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide By Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	None

10.10.1.6.4.10 ECC F_{2^m} Point Double - Affine Coordinates (ECC_F2M_DBL) Function

ECC_F2M_DBL computes the double (B + B) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 10-188. ECC_F2M_DBL function properties

Property	Notes
Mode value	0010_0000_000***0***_0000_1010 (output placed in B) 0010_0000_000***1***_0000_1010 (output placed in A)
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • A0, A1, A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod n • [B1, B2] = input point in affine coordinates • B3 = ignored
Output	P[B1, B2] (or P[A0, A1], if A output selected) = P[B1, B2] + P[B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field formed by N.
Side effects	A0, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

10.10.1.6.4.11 ECC F_{2^m} Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function

ECC_F2M_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_F2M_MUL_TEQ performs the same operation as ECC_F2M_MUL but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL will run faster than ECC_F2M_MUL_TEQ, but will never run slower.

Table 10-189. ECC_F2M_MUL and ECC_F2M_MUL_TEQ function properties

Property	Notes
Mode value for ECC_F2M_MUL	0010_0000_0**00*0***_0000_1011 (output placed in B)

Table continues on the next page...

Table 10-189. ECC_F2M_MUL and ECC_F2M_MUL_TEQ function properties (continued)

Property	Notes
	0010_0000_0**0**0***1***_0000_1011 (output placed in A)
Mode value for ECC_F2M_MUL_TEQ	0010_0000_0**1**1***0***_0000_1011 (output placed in B) 0010_0000_0**1**1***1***_0000_1011 (output placed in A)
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod n • B1 = ignored • B2 = ignored • B3 = ignored
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023 • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field.
Side effects	A0, A1 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For E = 0, this function returns a point at infinity (0,0).
- For E < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = -E). After the multiplication is complete, the formula -P= (x, x+y) can be used to compute the *y* coordinate of the effective final result; the *x* coordinate stays the same.

10.10.1.6.4.12 ECC F_{2^m} Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function

ECC_F2M_MUL_R2 computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. Since this function takes R^2 mod N as an additional input, ECC_F2M_MUL_R2 is more efficient than ECC_F2M_MUL, which

first must compute R^2 mod N before performing the multiplication. ECC_F2M_MUL_R2_TEQ performs the same operation as ECC_F2M_MUL_R2 but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL_R2 will run faster than ECC_F2M_MUL_R2_TEQ, but will never run slower.

Table 10-190. ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECC_F2M_MUL_R2	0011_0000_0**00*0***_0000_1011 (output placed in B) 0011_0000_0**00*1***_0000_1011 (output placed in A)
Mode value for ECC_F2M_MUL_R2_TEQ	0011_0000_0**10*0***_0000_1011 (output placed in B) 0011_0000_0**10*1***_0000_1011 (output placed in A)
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod *n* and m = degree of polynomial M • B1 = (f2m) R2 mod N, pre-computed as described in F2M_R2MODN (0Eh) • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equally size segments of A and B memory locations.
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023 • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field.
Side effects	A0, A1, A2, and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For k = 0, this function returns a point at infinity (0,0).
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x, x+y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

10.10.1.6.4.13 ECC F_{2m} Check Point (ECC_F2M_CHECK_POINT) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^2 + xy = x^3 + ax^2 + b$.

ECC_F2M_CHECK_POINT_R2 has R^2 mod N as an additional input, so it is more efficient than this function, which first must compute R^2 mod N before performing the operation.

Table 10-191. ECC_F2M_CHECK_POINT function properties

Property	Notes
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	• B1 = R^2 mod N, as described in F2M_R2MODN (0Eh) (Note that if the input point is invalid B1 and B2 may not be generated. Software should check the range and validity of x and y for the curve's equation before running this function.) • B2 = curve "b" parameter = c^4 mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

This function checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2 + xy = x^3 + ax^2 + b$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meanning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. The outputs will not be present if PIZ is set.

10.10.1.6.4.14 ECC F_{2m} Check Point, R^2 (ECC_F2M_CHECK_POINT_R2) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^2 + xy = x^3 + ax^2 + b$.

Since this function has R^2 mod N as an additional input, it is more efficient than ECC_F2M_CHECK_POINT, which first must compute R^2 mod N before performing the operation.

Table 10-192. ECC_F2M_CHECK_POINT_R2 function properties

Property	Notes
Input	• N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter • B1 = R^2 mod N, pre-computed as described in F2M_R2MODN (0Eh) • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	• B2 = curve "b" parameter = c^4 mod N (Note that if the input point is invalid B2 may not be generated. Software should check the range and validity of x and y for the curve's equation before running this function.)
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

This function checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2 + xy = x^3 + ax^2 + b$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meanning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. The output will not be present if PIZ is set.

10.10.1.6.4.15 ECM Modular Multiplication (ECM_MOD_MUL_X and ECM_MOD_MUL_X_TEQ) Function

ECM_MOD_MUL_X computes the scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. ECM_MOD_MUL_X_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3), constant for a given size of E. ECM_MOD_MUL_X in general will run faster than ECM_MOD_MUL_X_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, P2 = P3 + P1, where P1 is the input and P3 is the result. Though this computes just the x coordinate, there is enough information, with P1(y), to compute P3(y).

Table 10-193. ECM_MOD_MUL_X and ECM_MOD_MUL_X_TEQ function properties

Property	Notes
Mode value for ECM_MOD_MUL_X	0000_1000_0**00*0***_0100_1011 (output placed in B) 0000_1000_0**00*0***_0100_1011 (output placed in A)
Mode value for ECM_MOD_MUL_X_TEQ	0000_1000_0**10*0***_0100_1011 (output placed in B) 0000_1000_0**10*1***_0100_1011 (output placed in A)
Input	• N = modulus, a prime number. • E = scalar multiplier (k), any integer • [A0] = multiplicand, an input point's affine x coordinate • A2 = ignored • A3 = elliptic curve a24 parameter, that is, (A+2)/4 • B0 = ignored • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate. • N1 = R2 • A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R • A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor • A3 = a24R, the a24 input, with the Montgomery factor • B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R • B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor • B3 = X1R, the x input, with the Montgomery factor
Requirements	• Maximum modulus size = 128 bytes • The x in (A0) should be on the elliptic curve formed by (N, A3 and "B").
Side effects	All quadrants of A, B, and N are modified except N0.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N.
Flags set	PIZ is set if the result is the point at infinity.

10.10.1.6.4.16 ECM F_p Point Multiply, R^2 Mod N Input, Affine Coordinates (ECM_MOD_MUL_X_R2 and ECM_MOD_MUL_X_R2_TEQ) Function

ECM_MOD_MUL_X_R2 computes a scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. Since ECM_MOD_MUL_X_R2 has R^2 mod N as an additional input, this function is more

efficient than ECM_MOD_MUL_X, which first must compute R^2 mod N before performing the multiplication. ECM_MOD_MUL_X_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3), constant for a given size of E. ECM_MOD_MUL_X_R2 in general will run faster than ECM_MOD_MUL_X_R2_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, P2 = P3 + P1, where P1 is the input and P3 is the result. Though this computes just the x coordinate, there is enough information, with P1(y), to compute P3(y).

Table 10-194. ECM_MOD_MUL_X_R2 and ECM_MOD_MUL_X_R2_TEQ function properties

Property	Notes
Mode value for ECM_MOD_MUL_X_R2	0001_1000_0**00*0***_1000_1011 (output placed in B) 0001_1000_0**00*1***_1000_1011 (output placed in A)
Mode value for ECM_MOD_MUL_X_R2_TEQ	0001_1000_0**10*0***_1000_1011 (output placed in B) 0001_1000_0**10*1***_1000_1011 (output placed in A)
Input	• N = modulus, a prime number. • E = scalar multiplier (k), any integer • [A0] = multiplicand, an input point's affine x coordinate • A2 = ignored • A3 = elliptic curve a24 parameter, that is, (A+2)/4 • B0 = ignored • B1 = R2 mod N, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate. • N1 = R2 • A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R • A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor • A3 = a24R, the a24 input, with the Montgomery factor • B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R • B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor • B3 = X1R, the x input, with the Montgomery factor
Requirements	• Maximum modulus size = 128 bytes • The x in (A0) should be on the elliptic curve formed by (N, A3 and "B").
Side effects	All quadrants of A, B, and N are modified except N0.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N.
Flags set	PIZ is set if the result is the point at infinity.

10.10.1.6.4.17 ECT Modular Multiplication (ECT_MOD_MUL and ECT_MOD_MUL_TEQ) Function

ECT_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECT_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL in general will run faster than ECT_MOD_MUL_TEQ, but will never run slower.

Table 10-195. ECT_MOD_MUL and ECT_MOD_MUL_TEQ function properties

Property	Notes
Mode value for ECT_MOD_MUL	0000_0000_0**00*0***>_1000_1011 (output placed in B) 0000_0000_0**00*0***_1000_1011 (output placed in A)
Mode value for ECT_MOD_MUL_TEQ	0000_0000_0**10*0***_1000_1011 (output placed in B) 0000_0000_0**10*1***_1000_1011 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero • E = scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" *(or should this be "d")* parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0).
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(-k). After the computation is

complete, the formula -P= (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

10.10.1.6.4.18 ECT F_p Point Multiply, R^2 Mod N Input, Affine Coordinates (ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ) Function

ECT_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECT_MOD_MUL_R2 has R^2 mod N as an additional input, this function is more efficient than ECT_MOD_MUL, which first must compute R^2 mod N before performing the multiplication. ECT_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL_R2 in general will run faster than ECT_MOD_MUL_R2_TEQ, but will never run slower.

Table 10-196. ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECT_MOD_MUL_R2	0001_0000_0**00*0***_1000_1011 (output placed in B) 0001_0000_0**00*1***_1000_1011 (output placed in A)
Mode value for ECT_MOD_MUL_R2_TEQ	0001_0000_0**10*0***_1000_1011 (output placed in B) 0001_0000_0**10*1***_1000_1011 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = R2 mod N, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	• P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

10.10.1.6.4.19 ECT F_p Point Add, Affine Coordinates (ECT_MOD_ADD) Function

ECT_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 10-197. ECT_MOD_ADD function properties

Property	Notes
Mode value	0000_0000_000***0***_1000_1001 (output placed in B) 0000_0000_000***1***_1000_1001 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

10.10.1.6.4.20 ECT F_p Point Add, Affine Coordinates, R^2 Mod N Input (ECT_MOD_ADD_R2) Function

ECT_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECT_MOD_ADD_R2 has R^2 mod N as an additional input, this function is more efficient than ECT_MOD_ADD, which first must compute R^2 mod N before performing the addition.

Table 10-198. ECT_MOD_ADD_R2 function properties

Property	Notes
Mode value	0001_0000_000***0***_1000_1001 (output placed in B) 0001_0000_000***1***_1000_1001 (output placed in A)
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero • [A0, A1] = first addend point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend point in affine coordinates (x,y) • B3 = R2 (R^2 mod N) input
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

10.10.1.6.4.21 ECT F_p Check Point (ECT_MOD_CHECK_POINT) Function

ECT_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $ax^2 + y^2 = 1 + dx^2y^2$.

ECT_MOD_CHECK_POINT_R2 has R^2 mod N as an additional input, so it is more efficient than this function, which first must compute R^2 mod N before performing the operation.

Table 10-199. ECT_MOD_CHECK_POINT function properties

Property	Notes
Mode value	0000_0000_0000_1001_1100
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "d" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	• B1 = R^2 mod N, as described in MOD_R2MODN (0Eh)
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the "neutral point" (0, 1). GCD is set if the point is on the curve (but not the neutral point).

ECT_MOD_CHECK_POINT checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $ax^2 + y^2 = 1 + dx^2y^2$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

10.10.1.6.4.22 ECT F_p Check Point, R^2 (ECT_MOD_CHECK_POINT_R2) Function

ECT_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. x and y satisfy the equation $ax^2 + y^2 = 1 + dx^2y^2$.

Table 10-200. ECT_MOD_CHECK_POINT_R2 function properties

Property	Notes
Mode value	0001_0000_0000_1001_1100
Input	• N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "d" parameter • B1 = ignored • B2 = R^2 mod N, pre-computed as described in MOD_R2MODN (0Eh) • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.

Table continues on the next page...

Table 10-200. ECT_MOD_CHECK_POINT_R2 function properties (continued)

Property	Notes
Output	• B2 = R^2 mod N
Requirements	• Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	• Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the "neutral point" (0, 1).. GCD is set if the point is on the curve (but not the neutral point).

ECT_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $ax^2 + y^2 = 1 + dx^2y^2$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. Since ECT_MOD_CHECK_POINT_R2 has R^2 mod N as an additional input, this function is more efficient than ECT_MOD_CHECK_POINT, which first must compute R^2 mod N before performing the operation.

10.10.1.6.4.23 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)

These functions copy data from a PKHA register (or register quadrant) specified as a source, to another PKHA register (or register quadrant) specified as a destination. COPY NSZ copies the amount of data specified by the N Size register. COPY_SSZ copies the amount of data specified in the source register's size register. The source and destination are specified in the Mode Value. The source can be A, B or N. The destination can be A, B, E or N, but not the same as the source.

In a quadrant copy, when NSZ/SSZ exceeds the length of a quadrant, the copy will carry on into the next (higher-numbered) quadrant(s).

When the copy operation has completed, the destination register's size register will be updated to contain the number of bytes copied.

Table 10-201. COPY_NSZ and COPY_SSZ function properties

<table>
<tr><th>Property</th><th>Notes</th></tr>
<tr><td>Mode value</td><td>
<table>
<tr><th>Bits 19:17</th><th>Bits 16,11,10</th><th>Bits 9:8</th><th>Bits 7:6</th><th>Bits 5:0</th></tr>
<tr><th>Source Register</th><th>Destination Register</th><th>Source Segment</th><th>Destination Segment</th><th>Function Code</th></tr>
<tr><td>000 = A Register</td><td>000 = A Register</td><td>00 = Segment 0</td><td>00 = Segment 0</td><td rowspan="4">01_0000 = Copy_NSZ

01_0001 = Copy_SSZ</td></tr>
<tr><td>001 = B Register</td><td>001 = B Register</td><td>01 = Segment 1</td><td>01 = Segment 1</td></tr>
<tr><td>011 = N Register</td><td>011 = N Register</td><td>10 = Segment 2</td><td>10 = Segment 2</td></tr>
<tr><td></td><td>010 = E Register</td><td>11 = Segment 3</td><td>11 = Segment 3</td></tr>
<tr><td>other values reserved</td><td>other values reserved</td><td></td><td></td><td></td></tr>
</table>
1. If the destination register is E, the source and destination seqments must be 00b.
</td></tr>
<tr><td>Input</td><td>None</td></tr>
<tr><td>Output</td><td>None</td></tr>
<tr><td>Requirements</td><td>For Copy_NSZ, the N-size Register must contain a valid value.
For Copy_SSZ, the source register's size register must contain a valid value.</td></tr>
<tr><td>Side effects</td><td>The destination register's size register is updated to the number of bytes copied.</td></tr>
<tr><td>Errors reported</td><td>None</td></tr>
<tr><td>Flags set</td><td>None</td></tr>
</table>

1. If the destination register is E, the source and destination seqments must be 00b.

10.10.1.6.4.24 Right Shift A (R_SHIFT) function

Table 10-202. R_SHIFT function properties

Property	Notes
Mode value	0000_0000_000***0***_0001_1101 (output placed in B) 0000_0000_000***1***_0001_1101 (output placed in A)
Input	• A = Input value to be shifted. Bytes above A Size will be assumed to be zero, regardless of the contents of the PKHA A RAM. • B = Number of bit positions that the A RAM will be shifted (Only the least-significant two bytes are used. The upper bytes are ignored.)
Output	B (or A, if selected) = the contents of PKHA A RAM (with zeros substituted for bytes above A Size), right-shifted by the number of bit positions specified in the least-significant two bytes of PKHA B RAM, and zero-filled on the left.
Requirements	none
Side effects	B (or A) is modified.
Errors reported	none
Flags set	none

10.10.1.6.4.25 Compare A B (COMPARE) function

Table 10-203. COMPARE function properties

Property	Notes
Mode value	0000_0000_0000_0001_1110
Input	• A = Value to be compared • B = Value to be compared • A Size = the number of least-significant bits that will be compared
Output	None (other than flags)
Requirements	A Size must be >= B Size.
Side effects	none
Errors reported	If B Size > A Size a "B Size" error will be generated.
Flags set	• PKHA_GCD_ONE is set if B > A • PKHA_ZERO is set if B == A • no flag is set if B < A

10.10.1.6.4.26 Evaluate A (EVALUATE) function

Table 10-204. EVALUATE function properties

Property	Notes
Mode value	***SB***00_0000_000***0***_0001_1111 (output placed in B) ***SB***00_0000_000***1***_0001_1111 (do not modify B) • If the ***S*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0**sss**, where **sss** is the updated A Size. • If the ***B*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_**bbbb**, where **bbbb** is the updated number of bits in A. • If the ***S*** bit is set and the ***B*** bit is set, the A-Size DWord will be pushed before the number-of-bits-in-A Dword.
Input	• A = Value to be evaluated
Output	• A Size is updated with the number of least-significant non-zero bytes, i.e. the position of the most-significant non-zero byte (least-significant byte is byte-position 0). This evaluation considers only the bytes specified by the incoming value of A Size. This allows the incoming value of A Size to be set so that A0 will be evaluated, ignoring the values in A1, A2 and A3, or the incoming value could be set so that (A1,A0) will be evaluated, ignoring A2 and A3, or (A2,A1,A0) could be evaluated, ignoring A3. • If the mode value specifies that the output is to be placed in B, the updated value of A Size will be copied into the least-significant two bytes of B and B Size will be set to 2. If the "do not modify B" option is selected, the updated value of A Size will not be copied into B.
Requirements	none
Side effects	• A Size will be modified. • B Size may be modified. • One or two DWords mmay be pushed to the output FIFO.
Errors reported	none
Flags set	• PKHA_GCD_ONE is set if A == 1 • PKHA_ZERO is set if A == 0. A Size (and number of bits in A) will be set to 0. Note that this could cause an A Size error in a subsequent PKHA operation.

10.10.1.6.5 Special values for common ECC domains

Software can sometimes use the PKHA more effectively if the Montgomery Conversion Factor (R^2 mod N) is either provided or previously used to convert other inputs into Montgomery form. For convenience, the conversion factors for common ECC domains have been computed and published here. Some of the other domain values are provided to aid in definite identification of the domain, in the case that the name is not found or is not an exact match.

The following tables give these values for the *q* and *r* modulus values found in ECC domains. These associated Montgomery values are dependent upon the PKHA digit size (16, 32, 64, 128). These tables are for a PKHA with a 32-bit digit.

ECC F_{2^m} requires a *c* (also called *b'*) parameter for the elliptic curve in place of the *b* value. Table 10-206 provides these values in addition to the Montgomery values. The *b'* values are universal and do not change with PKHA digit size.

The following variable definitions apply to both tables. Variable names (*q, r, b, c*) follow the conventions of IEEE Std 1363.

Name

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org
- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

R

R is the Montgomery factor. Its value is 2^{SD}, where *D* is the PKHA digit size in bits, and *S* is the minimum number of digits needed to hold the modulus. As an example, for a modulus of nine bytes (72 bits), *R* would be

- 2^{80} for a PKHA with digit size of 16 bits

- 2^{96} for a PKHA with digit size of 32 bits
- 2^{128} for a PKHA with digit size of 64 or 128 bits

q

This is the *field-defining* value for the elliptic curve. For F_p curves, it is the prime number used as the modulus for all point arithmetic; it is named *p* in some other publications. For F_{2^m} curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

L

This is the number of bytes needed to hold *q* and each of its associated values: *R2modq*, *a*,*b*,*c*, the point coordinates *x* and *y*, the result of an ECDH key agreement, etc.

R2modq

This is R^2 *mod q*, the Montgomery Conversion Factor when *q* is the modulus.

r

This is the (usually prime) number which is the order of *G*, the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named *n* in some other publications.

N

This is the number of bytes needed to hold *r* and each of its associated values: *R2modr*, private keys, each of the two components of an ECDSA signature, etc.

R2modr

This is R^2 *mod r*, the Montgomery Conversion Factor when *r* is the modulus.

b / c (b')

b is the coefficient for the x^0 (ones) term in an F_{2^m} elliptic curve equation. Its relationship with *c* is $b = c^4$. *c* is sometimes referred to as *b'* in NXP documentation.

A / a24

a24 is the special value derived from the A coefficient for the y^2term in a Montgomery-form elliptic curve equation. Its relationship with *A* is *a24 = (A+2)/4*.

The domains in the table are ordered by size.

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits

Name	L	N	
	var		Value (hex, decimal, sums of powers)
secp112r1	14	14	
wtls6	q		0xDB7C2ABF62E35E668076BEAD208B 4451685225093714772084598273548427
	R2modq		0x00000000000000000000000009
	r		0xDB7C2ABF62E35E7628DFAC6561C5 4451685225093714776491891542548933

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modr		0xDA4A43AD7F34245D42B9C948C559
secp112r2	14	14	
	q		0xDB7C2ABF62E35E668076BEAD208B 4451685225093714772084598273548427
	R2modq		0x00000000000000000000000009
	r		0x36DF0AAFD8B8D7597CA10520D04B 1112921306273428674967732714786891
	R2modr		0x2049C67E5F79E8C06B7825955374
wtls8	14	15	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFDE7 5192296858534827628530496329219559 $2^{112} - 2^9 - 2^4 - 2^3 - 1$
	R2modq		0x00000000000000004667100000000
	r		0x0100000000000001ECEA551AD837E9 5192296858534827767273836114360297
	R2modr		0x00E074FD104C86569DB6C204A52932
secp128r1	16	16	
	q		0xFFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF 340282366762482138434845932244680310783 $2^{128} - 2^{97} - 1$
	R2modq		0x000000240000000400000000800000011
	r		0xFFFFFFFE0000000075A30D1B9038A115 340282366762482138443322565580356624661
	R2modr		0x71875047CDD8151626BC6448FADE9BED
secp128r2	16	16	
	q		0xFFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF 340282366762482138434845932244680310783 $2^{128} - 2^{97} - 1$
	R2modq		0x000000240000000400000000800000011
	r		0x3FFFFFFF7FFFFFFFBE0024720613B5A3 85070591690620534603955721926813660579
	R2modr		0x0EFCA409C09D126A99CD2E9404A3B434
secp160k1 ansix9p160k1	20	21	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73 1461501637330902918203684832716283019651637554291 $2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1$
	R2modq		0x0000000000000000000000010000A71A1B44BBA9

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	r		0x0100000000000000000001B8FA16DFAB9ACA16B6B3
			1461501637330902918203686915170869725397159163571
	R2modr		0x00CDCF2BABDFE35D2F4D8A8AAD0F8494330E687AAF
secp160r1	20	21	
ansix9p160r1	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFF
wtls7			1461501637330902918203684832716283019653785059327
			$2^{160} - 2^{31} - 1$
	R2modq		0x0000000000000000000000004000000100000001
	r		0x0100000000000000000001F4C8F927AED3CA752257
			1461501637330902918203687197606826779884643492439
	R2modr		0x00A0E626837A981E4B3CDC3854085E335F6744F8A4
secp160r2	20	21	
ansix9p160r2	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73
			1461501637330902918203684832716283019651637554291
			$2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^{9} - 2^{8} - 2^{7} - 2^{3} - 2^{2} - 1$
	R2modq		0x0000000000000000000000010000A71A1B44BBA9
	r		0x0100000000000000000000351EE786A818F3A1A16B
			1461501637330902918203685083571792140653176136043
	R2modr		0x0076E5A1814769EF9E8DD4D69E29AEB02AD8C126C7
wtls9	20	21	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC808F
			1461501637330902918203684832716283019655932313743
			$2^{160} - 2^{17} - 2^{16} - 2^{14} - 2^{13} - 2^{12} - 2^{11} - 2^{10} - 2^{9} - 2^{8} - 2^{6} - 2^{5} - 2^{4} - 1$
	R2modq		0x0000000000000000000000000000000C3C174FE1
	r		0x0100000000000000000001CDC98AE0E2DE574ABF33
			1461501637330902918203687013445034429194588307251
	R2modr		0x00CC3AB9A731EBB2AA87D1BED0AEF6B4CF5840D789
brainpoolP160r1	20	20	
	q		0xE95E4A5F737059DC60DFC7AD95B3D8139515620F
			1332297598440044874827085558802491743757193798159
	R2modq		0x6CF12F81C0CA7EF8FED717E0B333F8D625BC14FF
	r		0xE95E4A5F737059DC60DF5991D45029409E60FC09
			1332297598440044874827085038830181364212942568457
	R2modr		0x2BC73851FC9BE6F69E31FE16FC61D4351FDF90EA
brainpoolP160t1	20	20	
	q		0xE95E4A5F737059DC60DFC7AD95B3D8139515620F
			1332297598440044874827085558802491743757193798159

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modq		0x6CF12F81C0CA7EF8FED717E0B333F8D625BC14FF
	r		0xE95E4A5F737059DC60DF5991D45029409E60FC09 1332297598440044874827085038830181364212942568457
	R2modr		0x2BC73851FC9BE6F69E31FE16FC61D4351FDF90EA
P-192	24	24	
secp192r1 ansix9p192r1 prime192v1 ECPRGF192Random	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
	R2modq		0x0000000000000001000000000000000020000000000000001
	r		0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831 6277101735386680763835789423176059013767194773182842284081
	R2modr		0x28BE5677EA0581A24696EA5BBB3A6BEECE66BACCDEB35961
secp192k1 ansix9p192k1	24	24	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFEE37 6277101735386680763835789423207666416102355444459739541047 $2^{192} - 2^{32} - 2^{12} - 2^{8} - 2^{7} - 2^{6} - 2^{3} - 1$
	R2modq		0x000000000000000000000000000000100002392013C4FD1
	r		0xFFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D 6277101735386680763835789423061264271957123915200845512077
	R2modr		0x6A21191C2EC4B2B1F0F4F172195E97E2461C1989250F0702
prime192v2	24	24	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
	R2modq		0x0000000000000001000000000000000020000000000000001
	r		0xFFFFFFFFFFFFFFFFFFFFFFFE5FB1A724DC80418648D8DD31 6277101735386680763835789423078825936192100537584385056049
	R2modr		0xA4FEB8C277C030E139DA8CFB4E35E1F62814A261001BE8FF
prime192v3	24	24	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
	R2modq		0x0000000000000001000000000000000020000000000000001
	r		0xFFFFFFFFFFFFFFFFFFFFFFFF7A62D031C83F4294F640EC13 6277101735386680763835789423166314882687165660350679936019
	R2modr		0x45BCB42FF1CC05D0194D076B366D09BF0305982367330969
brainpoolP192r1	24	24	

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	q		0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297 4781668983906166242955001894344923773259119655253013193367
	R2modq		0xB6225126EED34F1033BF484602C3FE69E2474C6972C7B21A
	r		0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1 4781668983906166242955001894269038308119863659119834868929
	R2modr		0x98769B9CE772102BBF4AFD5DBF53AFF0B4727C80E407E8F8
brainpoolP192t1	24	24	
	q		0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297 4781668983906166242955001894344923773259119655253013193367
	R2modq		0xB6225126EED34F1033BF484602C3FE69E2474C6972C7B21A
	r		0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1 4781668983906166242955001894269038308119863659119834868929
	R2modr		0x98769B9CE772102BBF4AFD5DBF53AFF0B4727C80E407E8F8
P-224 secp224r1 ansix9p224r1 wtls12 ECPRGF224Random	28	28	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000001 26959946667150639794667015087019630673557916260026308143510066298881
	R2modq		0x00000000FFFFFFFFFFFFFFFFFFFFFFFE000000000000000000000001
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF16A2E0B8F03E13DD29455C5C2A3D 26959946667150639794667015087019625940457807714424391721682722368061
	R2modr		0xD4BAA4CF1822BC47B1E979616AD09D9197A545526BDAAE6C3AD01289
secp224k1 ansix9p224k1	28	29	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFE56D 26959946667150639794667015087019630673637144422540572481099315275117 $2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^{9} - 2^{7} - 2^{4} - 2^{1} - 1$
	R2modq		0x0000000000000000000000000000000000000010000352602C23069
	r		0x0100000000000000000000000001DCE8D2EC6184CAF0A971769FB1F7 26959946667150639794667015087019640346510327083120074548994958668279
	R2modr		0x00993FF72BB882BD88BBFF32E48BE0320816F60AF534CE24FBEC9FEAA0
brainpoolP224r1	28	28	
	q		0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF 22721622932454352787552537995910928073340732145944992304435472941311
	R2modq		0x0578FD592E6A6CE43FE8A2AA96AF774C43C20E727867CA8064DCD04F
	r		0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F 22721622932454352787552537995910923612567546342330757191396560966559
	R2modr		0x4A73A6563211A5611E9CAE249F24919B9399652CADDAF8AA486CA401
brainpoolP224t1	28	28	
	q		0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			22721622932454352787552537995910928073340732145944992304435472941311
	R2modq		0x0578FD592E6A6CE43FE8A2AA96AF774C43C20E727867CA8064DCD04F
	r		0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F 22721622932454352787552537995910923612567546342330757191396560966559
	R2modr		0x4A73A6563211A5611E9CAE249F24919B9399652CADDAF8AA486CA401
prime239v1	30	30	
	q		0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699839
	R2modq		0x0000000000000005000000000005FFFFFFFFFFFC0000000000800000000
	r		0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF9E5E9A9F5D9071FBD1522688909D0B 883423532389192164791648750360308884807550341691627752275345424702807307
	R2modr		0x2BE4B1BE15BDEB5DF3096A7BE4944FA0CB87DC9852A129052EC789ED615B
prime239v2	30	30	
	q		0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699839
	R2modq		0x0000000000000005000000000005FFFFFFFFFFFC0000000000800000000
	r		0x7FFFFFFFFFFFFFFFFFFFFFFF800000CFA7E8594377D414C03821BC582063 883423532389192164791648750360308886392687657546993855147765732451295331
	R2modr		0x76CF025EBF73DDE8A5D15F0C7C29FF23EED0AE5C096A0D32ABCB4B16B765
prime239v3	30	30	
	q		0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699839
	R2modq		0x0000000000000005000000000005FFFFFFFFFFFC0000000000800000000
	r		0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF975DEB41B3A6057C3C432146526551 883423532389192164791648750360308884771190369765922550517967171058034001
	R2modr		0x11500EB94E46F16737BEB7A266592D93C18845A5EB3F814C07B00EA6ACF5
P-256 secp256r1 ansix9p256r1 prime256v1 ECDSA-256 ecp_group_19 ECPRGF256Random	32	32	
	q		0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF 115792089210356248762697446949407573530086143415290314195533631308867097853951
	R2modq		0x00000004FFFFFFFDFFFFFFFFFFFFFFFEFFFFFFFBFFFFFFFF000000000000000 3
	r		0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			115792089210356248762697446949407573529996955224135760342422259061068512044369
	R2modr		0x66E12D94F3D956202845B2392B6BEC594699799C49BD6FA683244C95BE79EEA2
secp256k1	32	32	
ansix9p256k1	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F 115792089237316195423570985008687907853269984665640564039457584007908834671663 $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
	R2modq		0x000000000000000000000000000000000000000000000001000007A2000E90A1
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 115792089237316195423570985008687907852837564279074904382605163141518161494337
	R2modr		0x9D671CD581C69BC5E697F5E45BCD07C6741496C20E7CF878896CF21467D7D140
brainpoolP256r1	32	32	
	q		0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377 76884956397045344220809746629001649093037950200943055203735601445031516197751
	R2modq		0x4717AA21E5957FA8A1ECDACD6B1AC8075CCE4C26614D4F4D8CFEDF7BA6465B6C
	r		0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7 76884956397045344220809746629001649092737531784414529538755519063063536359079
	R2modr		0x0B25F1B9C32367629B7F25E76C815CB0F35D176A1134E4A0E1D8D8DE3312FCA6
brainpoolP256t1	32	32	
	q		0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377 76884956397045344220809746629001649093037950200943055203735601445031516197751
	R2modq		0x4717AA21E5957FA8A1ECDACD6B1AC8075CCE4C26614D4F4D8CFEDF7BA6465B6C
	r		0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7 76884956397045344220809746629001649092737531784414529538755519063063536359079
	R2modr		0x0B25F1B9C32367629B7F25E76C815CB0F35D176A1134E4A0E1D8D8DE3312FCA6

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
GOSTR3410-CryptoPro-A	32	32	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97 115792089237316195423570985008687907853269984665640564039457584007913129639319 $2^{256} - 2^9 - 2^6 - 2^5 - 2^3 - 1$
	R2modq		0x000000000000000000000000000000000000000000000000000000005CF11
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF6C611070995AD10045841B09B761B893 115792089237316195423570985008687907853073762908499243225378155805079068850323
	R2modr		0x551FE9CB451179DBF74885D08A3714C6FB07F8222E76DD529AC2D7858E79A469
GOSTR3410-CryptoPro-B	32	32	
	q		0x8000000000000000000000000000000000000000000000000000000000000C99 57896044618658097711785492504343953926634992332820282019728792003956564823193
	R2modq		0x000000000000000000000000000000000000000000000000000000027ACDC4
	r		0x8000000000000000000000000000015F700CFFF1A624E5E497161BCC8A198F 57896044618658097711785492504343953927102133160255826820068844496087732066703
	R2modr		0x09D1D2C4E50824664A2E7E2F6882CF102A3104A7EA43E85529B721F4E6CD7823
GOSTR3410-CryptoPro-C	32	32	
	q		0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B 70390085352083305199547718019018437841079516630045180471284346843705633502619
	R2modq		0x807A394EDE09765218630421284 9C07B1017BB39C2D346C5409973B4C427FCEA
	r		0x9B9F605F5A858107AB1EC85E6B41C8AA582CA3511EDDFB74F02F3A6598980BB9 70390085352083305199547718019018437840920882647164081035322601458352298396601
	R2modr		0x7AA61B49A49D4759C67E5D0EE96E8ED304FDA8694AFDA24BE94FAAB66ABA180E
brainpoolP320r1	40	40	
	q		0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27 176359332223916635416190984244601952088951277271951519277296041528864086880214981809550149990352 7

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modq		0xA259BA4A6C2D92525455A964E614D6D21F4C881F30C5B676C2478A8D906978EF994EE88A743B52F9
	r		0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311 1763593322239166354161909842446019520889512772717686063760686124016784784845843468355685258203921
	R2modr		0x31EC87C73200B14FE30D35244E6390FE86B330BCAF86C40991C3001BE0E16805679D29DF2513E4CD
brainpoolP320t1	40	40	
	q		0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27 1763593322239166354161909842446019520889512772719515192772960415288640868802149818095501499903527
	R2modq		0xA259BA4A6C2D92525455A964E614D6D21F4C881F30C5B676C2478A8D906978EF994EE88A743B52F9
	r		0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311 1763593322239166354161909842446019520889512772717686063760686124016784784845843468355685258203921
	R2modr		0x31EC87C73200B14FE30D35244E6390FE86B330BCAF86C40991C3001BE0E16805679D29DF2513E4CD
P-384 secp384r1 ansix9p384r1 ECDSA-384 ecp_group_20 ECPRGF384Random	48	48	
	q		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFF0000000000000000FFFFFFFF 394020061963944792122790401001436138050797392704654466679482934042457217714968703290472660882589380018616069731123 19
	R2modq		0x000000000000000000000000000000010000000200000000FFFFFFFE000000000000000200000000FFFFFFFE00000001
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC7634D81F4372DDF581A0DB248B0A77AECEC196ACCC52973 394020061963944792122790401001436138050797392704654466679469052796276593991132635693989563081522949135544336539426 43
	R2modr		0x0C84EE012B39BF213FB05B7A28266895D40D49174AAB1CC5BC3E483AFCB82947FF3D81E5DF1AA4192D319B2419B409A9
brainpoolP384r1	48	48	
	q		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53 216592707701193161730692368423326049797961163870176486000816185038210899340259618222365619828445340884407084179733 31
	R2modq		0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC9940899535283343D7FD965087CEFFF40B64BDE
	r		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			21659270770119316173069236842332604979796116387017648600075645274821611501358515537962695117368903252229601718723941
	R2modr		0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688A37264E202F2B6B6EAC4ED3A2DE771C8E
brainpoolP384t1	48	48	
	q		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53 21659270770119316173069236842332604979796116387017648600081618503821089934025961822236561982844534088440708417973331
	R2modq		0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC9940899535283343D7FD965087CEFFF40B64BDE
	r		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565 21659270770119316173069236842332604979796116387017648600075645274821611501358515537962695117368903252229601718723941
	R2modr		0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688A37264E202F2B6B6EAC4ED3A2DE771C8E
brainpoolP512r1	64	64	
	q		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3 8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947
	R2modq		0x3C4C9D05A9FF6450202E19402056EECCA16DAA5FD42BFF8319486FD8D5898057E0C19A7783514A2553B7F9BC905AFFD3793FB1302715790549AD144A6158F205
	r		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069 8948962207650232551656602815159153422162609644098354511344597187200057010413418528378981730643524959857451398370029280583094215613882043973354392115544169
	R2modr		0xA794586A718407B095DF1B4C194B2E56723C37A22F16BBDFD7F9CC263B790DE3A6F230C72F0207E83EC64BD033B7627F0886B75895283DDDD2A3681ECDA81671
brainpoolP512t1	64	64	
	q		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3 8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947
	R2modq		0x3C4C9D05A9FF6450202E19402056EECCA16DAA5FD42BFF8319486FD8D5898057E0C19A7783514A2553B7F9BC905AFFD3793FB1302715790549AD144A6158F205

Table continues on the next page...

Table 10-205. Special Values for common ECC F_p domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	r		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069 8948962207650232551656602815159153422162609644098354511344597187200057010413418528378981730643524959857451398370029280583094215613882043973354392115544169
	R2modr		0xA794586A718407B095DF1B4C194B2E56723C37A22F16BBDFD7F9CC263B790DE3A6F230C72F0207E83EC64BD033B7627F0886B75895283DDDD2A3681ECDA81671
P-521 secp521r1 ansix9p521r1 ECDSA-521 ecp_group_21 ECPRGF521Random	66	66	
	q		0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151 $2^{521} - 1$
	R2modq		0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000400000000000
	r		0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA51868783BF2F966B7FCC0148F709A5D03BB5C9B8899C47AEBB6FB71E91386409 6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449
	R2modr		0x019A5B5A3AFE8C44383D2D8E03D1492D0D455BCC6D61A8E567BCCFF3D142B7756E3A4FB35B72D34027055D4DD6D30791D9DC18354A564374A6421163115A61C64CA7

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits

Name	L	N	
	var		Value (hex, decimal, sums of powers)
sect113r1 wtls4	15	15	
	q		0x020000000000000000000000000201 $x^{113} + x^9 + 1$
	R2modq		0x000000000000000001000040000000
	b c		0xE8BEE4D3E2260744188BE0E9C723 0x0173E834AF28EC76CB83BD8DFEB2D5
	r		0x0100000000000000D9CCEC8A39E56F 5192296858534827689835882578830703

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modr		0x002D02609ABE76F866BDCE5B3F9BCC
sect113r2	15	15	
	q		0x020000000000000000000000000201 $x^{113} + x^9 + 1$
	R2modq		0x00000000000000000001000040000000
	b		0x95E9A9EC9B297BD4BF36E059184F
	c		0x0054D9F03957174A32329167D7FE71
	r		0x01000000000000000108789B2496AF93 519229685853482770297249790995240 3
	R2modr		0x00471CB662E29CB41ABC888E16FF49
wtls1	15	14	
	q		0x020000000000000000000000000201 $x^{113} + x^9 + 1$
	R2modq		0x00000000000000000001000040000000
	b		0x01
	c		0x00000000000000000000000000000001
	r		0xFFFFFFFFFFFFFFFFDBF91AF6DEA73 519229685853482762789670383346750 7
	R2modr		0x9A1AB7E0A60C212FBD48A8239130
sect131r1	17	17	
	q		0x080000000000000000000000000000010D $x^{131} + x^8 + x^3 + x^2 + 1$
	R2modq		0x00000000000000040144000000000000
	b		0x0217C05610884B63B9C6C7291678F9D341
	c		0x03DB89B405E491160E3B2F07B0CE20B37E
	r		0x040000000000000000 23123953A9464B54D 136112946768375385389393275568536556065 3
	R2modr		0x00739BBCD15B208AC45847F42ED438E023
sect131r2	17	17	
	q		0x080000000000000000000000000000010D $x^{131} + x^8 + x^3 + x^2 + 1$
	R2modq		0x00000000000000040144000000000000
	b		0x04B8266A46C55657AC734CE38F018F2192
	c		0x07CBB9920D71A48E099C38D71DA6490EB1
	r		0x040000000000000000 16954A233049BA98F 136112946768375385387953504341281 2 867983
	R2modr		0x01BB89631FCB716E50598B58B5058E0389

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
Oakley 3	20	-	
	q		0x080000000000000000000000400000000000000001 $x^{155} + x^{62} + 1$
	R2modq		0x0000004000000000000000000000000000000400
	b		0x07338F
	c		0x00311000000223A000C4474000088E8000111D1D
B-163 ansix9t163r2 sect163r2 EC2NGF163Random	21	21	
	q		0x0800000000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
	R2modq		0x00000000000000000000000141040000000000000000
	b		0x020A601907B8C953CA1481EB10512F78744A3205FD
	c		0x072C4E1EF7CB2F3A035D33104294159609138BB404
	r		0x040000000000000000000292FE77E70C12A4234C33 584600654932361167281474244287639068925684320158 7
	R2modr		0x003488BE6C9C552CFE775F73CFB60B416A9AA88652
K-163 ansix9t163k1 sect163k1 EC2NGF163Koblitz wtls3	21	21	
	q		0x0800000000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
	R2modq		0x00000000000000000000000141040000000000000000
	b		0x01
	c		0x0000000000000000000000000000000000000001
	r		0x040000000000000000000020108A2E0CC0D99F8A5EF 584600654932361167281474175359844834832911857 4063
	R2modr		0x01719E20D16A34F5053B1368AE089C83FBAA63410E
sect163r1 ansix9t163r1	21	21	
	q		0x0800000000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
	R2modq		0x00000000000000000000000141040000000000000000
	b		0x0713612DCDDCB40AAB946BDA29CA91F73AF958AFD9
	c		0x05ED403ED58EB45B1CCECA0F4F61655549861BE052
	r		0x03FFFFFFFFFFFFFFFFFFF48AAB689C29CA710279B 584600654932361167281473846509879898130442041129 1
	R2modr		0x03F45AD7608E554A90299358F3719D4236333FE8B2
wtls5	21	21	
	q		0x0800000000000000000000000000000000000000107 $x^{163} + x^8 + x^2 + x^1 + 1$
	R2modq		0x00000000000000000000000400540000000000000000

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	b		0xC9517D06D5240D3CFF38C74B20B6CD4D6F9DD4D9
	c		0x0453E1E4B7291F5C2D53CE18483F007081E7EA26EC
	r		0x040000000000000000000001E60FC8821CC74DAEAFC1
			584600654932361167281474162622639205657383263840 1
	R2modr		0x02704CFBABEA28A831BAD35BCAA440A89884D1FA9B
Oakley 4	24	-	
	q		0x020000000000000000000000000000020000000000000001
			$x^{185} + x^{69} + 1$
	R2modq		0x000000000100000000000000000000000000000000004000
	b		0x1EE9
	c		0x000000000000300000018000C00C000000638030001C0009
sect193r1	25	25	
ansix9t193r1	q		0x020000000000000000000000000000000000000000008001
			$x^{193} + x^{15} + 1$
	R2modq		0x000000000000000000000000010000000400000000000000
	b		0xFDFB49BFE6C3A89FACADAA7A1E5BBC7CC1C2E5D831478814
	c		0x0167B35EB4313F263D0F7A3D5036F0A0A3C980D40E5A053ED2
	r		0x01000000000000000000000000C7F34A778F443ACC920EBA49
			6277101735386680763835789423269548053691575186051040197193
	R2modr		0x009F0A6812CD5A0961578029E866525B193ED6F556637F68CD
sect193r2	25	25	
ansix9t193r2	q		0x020000000000000000000000000000000000000000008001
			$x^{193} + x^{15} + 1$
	R2modq		0x000000000000000000000000010000000400000000000000
	b		0xC9BB9E8927D4D64C377E2AB2856A5B16E3EFB7F61D4316AE
	c		0x006989FE6BFE30EDDC3244269F3AAD18D66CF3DB3E3302FAA8
	r		0x01000000000000000000000015AAB561B005413CCD4EE99D5
			6277101735386680763835789423314955362437298222279840143829
	R2modr		0x00B24356750478EA3C6D96955F00208DD023F898087B1BF123
B-233	30	30	
sect233r1	q		0x020000000000000000000000000000000000000004000000000000000001
ansix9t233r1			$x^{233} + x^{74} + 1$
EC2NGF233Random	R2modq		0x000000000004000000000000000000000000000000000400000000000
wtls11	b		0x66647EDE6C332C7F8C0923BB58213B333B20E9CE4281FE115F7D8F90AD
	c		0x0007D5EF4389DFF11ECDBA39C30970D3CE35CEBBA58473F64B4DC0F2686C
	r		0x0100000000000000000000000013E974E72F8A6922031D2603CFE0D7

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			690174634679056378743475586227702555583981273734501355537938363448546 3
	R2modr		0x006AB044AA57CDD6D0CC9138B004578CD5EFE7E89545CDAA1BA1C26DD4D1
K-233	30	29	
sect233k1 ansix9t233k1 EC2NGF233Koblitz wtls10	q		0x020000000000000000000000000000000000000004000000000000000001 $x^{233} + x^{74} + 1$
	R2modq		0x000000000004000000000000000000000000000000000400000000000
	b		0x01
	c		0x000000000000000000000000000000000000000000000000000000000000001
	r		0x800000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF 345087317339528189371737793113851276057094098886225212632808702474134 3
	R2modr		0x59BEBED80293C813EEB5B58A0AF7E3EB91DB9A5B861710AC1009468BB6
sect239k1 ansix9t239k1	30	30	
	q		0x800000000000000000040000000000000000000000000000000000000001 $x^{239} + x^{158} + 1$
	R2modq		0x000000000001000000000000000000800000000000000000440000000
	b		0x01
	c		0x000000000000000000000000000000000000000000000000000000000000001
	r		0x2000000000000000000000000005A79FEC67CB6E91F1C1DA800E478A5 220855883097298041197912187592864814948216561321709848887480219215362 213
	R2modr		0x183E8C975E5EA68E203395FBEC1187B0F40DFFCA2CE64F17F77925590A73
B-283 sect283r1 ansix9t283r1 EC2NGF283Random	36	36	
	q		0x0800000000000000000000000000000000000000000000000000000000000000000 010A1 $x^{283} + x^{12} + x^7 + x^5 + 1$
	R2modq		0x0000000000000000000000000000000000000000000000000000000000000004011 00400
	b		0x027B680AC8B8596DA5A4AF8A19A0303FCA97FD7645309FA2A581485AF6263E3 13B79A2F5
	c		0x03D8C93D3B0EA81D9294034D7EE3135D0AC5FC8D9CB0276F7211F880F0D81C A4C6E87B38
	r		0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEF90399660FC938A90165B042A7 CEFADB307 777067556890291628367784762729407562656962592437690488910919652677004 4277787378692871
	R2modr		0x0299ADD3FE013DB2E23755FAA9545A49222D8461643773D41D288BCA1EBB695 767D7CA78
K-283	36	36	

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
sect283k1 ansix9t283k1 EC2NGF283Koblitz	q		0x08000000000000000000000000000000000000000000000000000000000000000010A1 $x^{283} + x^{12} + x^7 + x^5 + 1$
	R2modq		0x0000000000000000000000000000000000000000000000000000000000000401100400
	b c		0x01 0x0000000000000000000000000000000000000000000000000000000000000000000001
	r		0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9AE2ED07577265DFF7F94451E061E163C61 38853377844514581418389238136470378132848117337930613242958749975298158297044226038 73
	R2modr		0x00868DB4F6424B4B297831F5EBA11AE2B1AC177F33C6C133859892757E41B7CAE8927BDB
B-409 sect409r1 ansix9t409r1 EC2NGF409Random	52	52	
	q		0x0200000000000000000000000000000000000000000000000000000000000000000000000000000080000000000000000000001 $x^{409} + x^{87} + 1$
	R2modq		0x000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000004000
	b c		0x21A5C2C8EE9FEB5C4B9A753B7B476B7FD6422EF1F3DD674761FA99D6AC27C8A9A197B272822F6CD57A55AA4F50AE317B13545F 0x0149B8B7BEBD9B63653EF1CD8C6A5DD105A2AAAC36FE2EAE43CF28CE1CB7C830C1ECDBFA413AB07FE35A57811AE4F88D30AC63FB
	r		0x010000000000000000000000000000000000000000000000001E2AAD6A612F33307BE5FA47C3C9E052F838164CD37D9A21173 661055968790248598951915308032771039828404682964281219284648798304157774827374805208143723762179110965979867288366567526771
	R2modr		0x0007C24B27E70A941D4738F415F186A66A9FF8783C798E99D4A152BF0CE0C0CC273CDA3D70AA82C43A336EFBE1479034DB8EF936
K-409 sect409k1 ansix9t409k1 EC2NGF409Koblitz	52	51	
	q		0x0200000000000000000000000000000000000000000000000000000000000000000000000000000080000000000000000000001 $x^{409} + x^{87} + 1$
	R2modq		0x000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000004000
	b c		0x01 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
	r		0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE5F83B2D4EA20400EC4557D5ED3E3E7CA5B4B5C83B8E01E5FCF

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			3305279843951242994759576540163855199142023414821406096423243950228807112892491910506732584577774580140963665906177313586 71
	R2modr		0x50DC9B805D4BEBA0701EDA0D529DAD74A3ED9914801EFC3F5D0760180600F3725B714A1C6E7B3C68A06DF3709E9354226F8D6C
B-571 sect571r1 ansix9t571r1 EC2NGF571Random	72	72	
	q		0x0800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000425 $x^{571} + x^{10} + x^5 + x^2 + 1$
	R2modq		0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000040104400
	b c		0x02F40E7E2221F295DE297117B7F3D62F5C6A97FFCB8CEFF1CD6BA8CE4A9A18AD84FFABBD8EFA59332BE7AD6756A66E294AFD185A78FF12AA520E4DE739BACA0C7FFEFF7F2955727A 0x06395DB22AB594B1868CED952578B6539FABA69406D9B2986123A185C85832E25FD5B63833D51442ABF1A9C05FF0ECBD88D7F77997F4DC9156AAF1CE08164686DDFF75116FBC9A7A
	r		0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE661CE18FF55987308059B186823851EC7DD9CA1161DE93D5174D66E8382E9BB2FE84E47 3864537523017258344695351890931987344298927329706434998657235251451519142289560424536143999389415773083133881121926944486246872462816813070234528288303332411393191105285703
	R2modr		0x00780C1005944C99C498CDB275BF7CCC389C0853C856C10F3786A7DCF3AA9AE196A3FB16F1DE5AF21B1318667E55C15B9A8ABF1B469BD13D57BB95B60B677DBCAA35B843B87069F9
K-571 sect571k1 ansix9t571k1 EC2NGF571Koblitz	72	72	
	q		0x0800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000425 $x^{571} + x^{10} + x^5 + x^2 + 1$
	R2modq		0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000040104400
	b c		0x01 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
	r		0x020000000000000000000000000000000000000000000000000000000000000000000131850E1F19A63E4B391A8DB917F4138B630D84BE5D639381E91DEB45CFE778F637C1001 1932268761508629172347675945465993672149463664853217499328617625725759571144780212268133978522706711834706712800825351461273674974066617311929682421617092503555733685276673

Table continues on the next page...

Table 10-206. Special Values for common ECC F_{2^m} domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modr		0x019433720D8C7057F7F3F824CCB3E09071584DD65C1437B2406C8210EEF4949565D35EE4FE01AAEE96D0DC137749D25AA49C07F63F829BF85960C535AA90F11DDEC4B62B18F2E26D

For these Montgomery domains, some functions require *a24* instead of *A*. The PKHA requires that a24 be (A+2)/4.

Table 10-207. Special Values for common ECM MOD (Montgomery curves) domains when PKHA digit size is 32 bits

Name	L	N	
	var		Value (hex, decimal, sums of powers)
Curve25519	32	32	
	q		0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFED 2^{255}-19
	R2modq		0x0000000000000000000000000000000000000000000000000000000000000005A4
	A A24		0x076D06 486662 0x01DB42 121666
	r		0x1000000000000000000000000000000014DEF9DEA2F79CD65812631A5CF5D3ED 7237005577332262213973186563042994240857116359379907606001950938285454250989
	R2modr		0x0399411B7C309A3DCEEC73D217F5BE65D00E1BA768859347A40611E3449C0F01

Table 10-208. Special Values for common ECT MOD (Edwards curves) domains when PKHA digit size is 32 bits

Name	L	N	
	var		Value (hex, decimal, sums of powers)
edwards25519	32	32	
	q		0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFED 2^{255}-19
	R2modq		0x0000000000000000000000000000000000000000000000000000000000000005A4
	r		0x1000000000000000000000000000000014DEF9DEA2F79CD65812631A5CF5D3ED

Table continues on the next page...

Table 10-208. Special Values for common ECT MOD (Edwards curves) domains when PKHA digit size is 32 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			723700557733226221397318656304299424085711635937990760600195093828545 4250989
	R2modr		0x0399411B7C309A3DCEEC73D217F5BE65D00E1BA768859347A40611E3449C0F0 1

10.10.2 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.

10.10.2.1 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 10-209. 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.

10.10.2.2 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.

10.10.2.3 AFHA use of the Key Register

The AFHA uses the Key Register as follows:

- The Key Register contains the 1-to-32 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 32, produces an AFHA key-size error.
- The Key Register and key size registers are ignored in UPDATE or FINALIZE modes.

10.10.2.4 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.

10.10.2.5 Save and restore operations in AFHA S-box and AFHA context data

The AFHA S-box is a 256-byte array used during encryption or decryption, the contents of which changes with each byte of message being processed. The context pointers I and J are each a single byte of data that also changes with each byte of message being processed. If a descriptor is used to ARC4-encrypt or decrypt only a part of a message, and then a subsequent descriptor is used to process another part of the same message, the S-box and context pointers within AFHA must be saved by the first descriptor and restored by the subsequent descriptor.

At the end of an INIT or UPDATE mode operation, the S-box can be saved to memory (automatically encrypted using the JDKEK or TDKEK), with the data from S-box being read out in order, first byte to last. If the S-box is saved, the I and J pointers are also saved from the Context Register at the same time, as described in the AFHA Context Register section, above.

At the start of UPDATE or FINALIZE mode operations, the S-box 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 into the Context Register at the same time.

10.10.2.5.1 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.

10.10.2.6 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, 258 bytes must be loaded. Attempting to load more or less than 258 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.

10.10.3 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, CFB8, and OFB modes as well as key parity checking in compliance with the DES specification. DES is controlled from the class 1 CHA registers.

10.10.3.1 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), 8-bit cipher feedback (CFB8), and output feedback (OFB), described as follows:
  - ECB (0x20h) 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 (0x10h) 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.
  - CFB8 (0x30h) mode is a confidentiality mode that features the feedback of successive 8-bit 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 CFB8 mode requires an IV as the initial input-block.
  - OFB (0x40h) 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 (0x80h) in the AAI field.
- The algorithm field (ALG) must be programmed to DES (0x20h) or 3DES (0x21h).

10.10.3.2 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.

10.10.3.3 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.

10.10.3.4 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.

10.10.3.5 DESA Context Register

The DESA uses the Context Register as follows:

- For CBC, OFB, and CFB8 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 eight bytes of the Context Register to hold the beginning and final IV value for the CBC, OFB, and CFB8 modes. The bits are assigned as follows: Context DWord0: IV = desa_context[63:0]

10.10.3.6 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.

10.10.4 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 deterministic random-bit generator (DRBG) 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.

10.10.4.1 RNG features summary

The RNG module includes these distinctive features:

- Complete implementation of DRBG_Hash (SP800-90A) using SHA-256.
- Support for two state handles.
- Built-in entropy source conforming to SP800-90B and BSI AIS/31.
- Integrated entropy source for instantiating and re-seeding the DRBG.
- The RNG may be accessed through a register interface for test purposes.
- 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 using job descriptor.
- Random numbers automatically supplied as padding when needed by protocols.

10.10.4.2 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 deterministic random-bit generator (DRBG). TRNG creates real entropy (seed generation) and DRBG generates cryptographically strong data using this entropy (random-number generation).

10.10.4.2.1 RNG state handles

The RNG in CAAM implements 2 state handles. Each state handle:

- is a completely independent virtual RNG.
- may be independently instantiated in deterministic or nondeterministic mode, and with or without prediction resistance
- is independently seeded (or reseeded) with independent entropy

Thereafter, each state handle maintains an independent context for the RNG's deterministic random bit generator (DRBG).

Note that the JDKEK, TDKEK, TDSK and ZMK (if the ZMK is set for hardware programming) 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.

10.10.4.2.2 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. 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 10-210. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK

Descriptor	Value	Execution
Descriptor to instantiate RNG State Handle 0 in deterministic (test) mode **NOTE:** This descriptor would be executed prior to running tests on the RNG.	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 **NOTE:** This descriptor would be executed after running tests on the RNG.	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

Table continues on the next page...

Table 10-210. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK (continued)

Descriptor	Value	Execution
Descriptor to instantiate RNG State Handle 0 in non-deterministic (normal) mode with a personalization string and then load the JDKEK, TDKEK, and TDSK registers **NOTE:** This descriptor would be executed to start normal operation of the RNG and also initialize JDKEK, TDKEK and TDSK.	B080 0008h	HEADER command, indicating a descriptor with a length of eight 32-bit words
	12A0 0004h	LOAD Command with 4 bytes of immediate data; destination is Class 1 Context register
	nnnn nnnnh	4 bytes of immediate data (Personalization String) Different values (e.g. chip ID) should be used to ensure that the RNG on each chip is initialized differently.
	8250 4004h	OPERATION command, Class 1, RNG, Instantiate with Personalization String, 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 interrrupt and returns the RNG to idle.)
	8250 1000h	OPERATION command, Class 1, RNG, Secure Key (e.g. load JDKEK, TDKEK and TDSK registers), 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.		
Descriptor to instantiate RNG State Handle 0 non-deterministic (normal) mode with a personalization string **NOTE:** This descriptor would be executed to start normal operation of the RNG, but without initializing JDKEK, TDKEK and TDSK.	B080 0004h	HEADER command, indicating a descriptor with a length of four 32-bit words
	12A0 0004h	LOAD Command with 4 bytes of immediate data; destination is Class 1 Context register
	nnnn nnnnh	4 bytes of immediate data (Personalization String) Different values (e.g. chip ID) should be used to ensure that the RNG on each chip is initialized differently.
	8250 4004h	OPERATION command, Class 1, RNG, Instantiate with Personalization String, Non-Test Mode

10.10.4.3 RNG operations

RNG operations are performed by appropriately setting the Algorithm State (AS) field of the OPERATION command.

Table 10-211. 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

Table continues on the next page...

Table 10-211. RNG Operations (continued)

Value of AS	Name	Function
		requested data. The TST bit value must match the deterministic/nondeterministic mode of the selected state handle, else a test error is generated.[1] 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	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. • TST bit = 0. Nondeterministic instantiation. When instantiating a state handle in nondeterministic (normal) mode, the state handle is seeded with 512 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. • 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 256 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.
10	State-handle reseed operation	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. • For a state handle in nondeterministic mode, the DRNG is seeded with 512 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register. • 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.
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.

1. There is one exception to this rule. A test error is not 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.

10.10.4.4 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)
  - 256-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.

10.10.4.5 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, or to supply an optional 256 bits of additional input when reseeding a state handle or generating random data.

10.10.4.6 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 number of ***generate*** requests.
- The RNG uses a default reseed value of 10,000,000 requests. This means that 10,000,000 ***generate*** requests are processed before an automatic reseed operation occurs. For a system with the clock speed between 133MHz - 400MHz, the reseed happens between 3-20 seconds if RNG operations are being processed at the maximum rate.
- The Data Size Register holds 32 bits so the user can specify a larger or smaller value. If the user does not specify a reseed interval, the default value is used.

10.10.5 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.

10.10.5.1 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 Algorithm field (ALG) must be programmed to MD5, SHA-1, SHA-224, or SHA-256.
- 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:

Table 10-212. Mode Register[AS] operation selections in MDHA

Operation	Description
INIT	The hashing algorithm is initialized with the chaining variables and then hashing begins. Input data must be a non-zero multiple 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 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.

- The Additional Algorithm Information field (AAI) 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, SMAC, and HMAC with Derived HMAC Key.
  - The HMAC mode is defined by FIPS 198-1. This can be performed with any of the hashing algorithms.
  - The SMAC mode is defined by the SSL 3.0 specification. This can be performed with MD5 or SHA-1 hashing algorithm only.
  - The HMAC with Derived HMAC Key performs the parts of the HMAC algorithm that are not part of generating the Derived HMAC Key. This optimization saves two block computations of the underlying hash function. See

Using the MDHA Key Register with Derived HMAC Keys for more information.

NOTE

For HMAC and SMAC, the MD5 Key cannot be shared between DECOs until the donor MDHA is done. As a result, if using an MD5 key in a shared descriptor, sharing must be set to NEVER, WAIT or SERIAL, and sharing cannot be permitted to proceed until MDHA is done. For more information on sharing, please refer to Table 10-8.

10.10.5.2 MDHA use of the Key Register

The MDHA uses the Key Register as follows:

- The Key Register set is only used when a non-0 value in the Mode Register AAI field is specified.
- These registers either hold the HMAC key or the Derived HMAC Key.
- The two components of the Derived HMAC Key are each the size of the digest that is defined by the specified algorithm, except for SHA-224, which is 32 bytes.

10.10.5.2.1 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.

10.10.5.2.2 Using the MDHA Key Register with Derived HMAC Keys

The HMAC function uses an HMAC key per the following equation:

HMAC(Key,Message) = Hash[(Key ⊕ OPAD) || HASH((Key ⊕ IPAD) || Message)],

where "IPAD" is the constant byte 36h repeated 64 times for MD5, SHA-1, SHA-224 and SHA-256, and "OPAD" is the constant byte 5Ch repeated similarly. "Key" is the HMAC Key, and "HASH" is the chosen hashing function (for example, SHA-256).

10.10.5.2.2.1 Definition and function of Derived HMAC Keys

To improve performance, CAAM permits the use of Derived HMAC Keys. Computing the values Hash(Key ⊕ IPAD) and Hash(Key ⊕ OPAD) each require MDHA to perform one block of the underlying hash computation. By performing these computations once,

subsequent HMAC computations can save two blocks of hash computation. As a result, the Key Command has an option to signal MDHA that the key being loaded is a derived key.

10.10.5.2.2.2 Process flow when using the Key Register with Derived HMAC Keys

When MDHA runs, it turns a key into a Derived HMAC Key. MDHA writes this derived key form back to the Class 2 Key Register. Because the derived key is required every time, it saves time to create it once and then reuse it rather than starting with the key again. However, the derived key does not appear in the Key Register contiguously. To make the hardware much simpler, the half of the derived key generated with IPAD appears at the start of the Key Register and the half of the derived key generated with OPAD starts at the midpoint. So, for MD5, between these two 16-byte portions of the key there can be 16 bytes of null data. Using the Derived HMAC KEY avoids the need to load or store the null data.

10.10.5.2.2.3 Using padding with the Derived HMAC Key to align with storage

When doing a FIFO STORE of a Derived HMAC Key, the user provides a length equivalent to the sum of the bytes in the Derived HMAC Key itself. 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 Derived HMAC Key (size would be 40), there must be "padding" of 4 bytes following each portion of the derived key. That is: {20 bytes of the IPAD half of the derived key, 4 bytes of pad, 20 bytes of the OPAD half of the derived key, 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 derived key is stored, where the extra padding is used to pad each portion to an 8-byte boundary so that they can be handled separately.

10.10.5.2.2.4 Length of a Derived HMAC Key

Because the Derived HMAC Key key consists of two blocks of material processed by the selected hash algorithm, the length of a Derived HMAC Key is twice the length of the hash algorithm's running digest (note exception below). Storage of the Derived HMAC Key in the Class 2 Key Register, however, is such that the IPAD half of the derived key is at offset zero, and the OPAD half of the derived key is at offset 32.

10.10.5.2.2.5 Loading/storing a Derived HMAC Key with a KEY command

A Derived HMAC Key may be loaded either in encrypted or in unencrypted form. The DECO command to load a Derived HMAC Key is the KEY command with the KDEST field set to 3h. A Derived HMAC 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 Derived HMAC Key in external memory is 64 bytes. In addition, the length for a KEY command associated with a SHA-224 Derived HMAC Key must be 64. A Derived HMAC Key that CAAM has generated has also been encrypted, and for SHA-1, has been padded with 8 additional bytes.

10.10.5.2.2.6 Loading/storing a Derived HMAC Key with a FIFO STORE command

The DECO command to store a Derived HMAC Key is the FIFO STORE command, with output-data type set to 16 or 26 (17 or 27 to encrypt using the TDKEK). Generating a Derived HMAC Key in this fashion results in the key being encrypted with the JDKEK or TDKEK. Note that the length of an encrypted Derived HMAC 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 Derived HMAC Key is always longer, because CAAM must add 8 bytes of padding before the pre-encrypted 40 bytes of actual Derived HMAC Key can be encrypted.

10.10.5.2.2.7 Sizes of Derived HMAC Keys

This table describes the different sizes of derived HMAC keys depending on how they were generated.

Table 10-213. Sizes of derived HMAC keys

Hash algorithm	Final digest size	Running digest size	Software-generated derived key size	AES-ECB encrypted derived key size	AES-CCM encrypted derived 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

10.10.5.2.2.8 Storing an HMAC-SHA-1 Derived Key in Memory

This figure is an example of how software would store a derived key for HMAC-SHA-1 in memory, given the derived HMAC key.

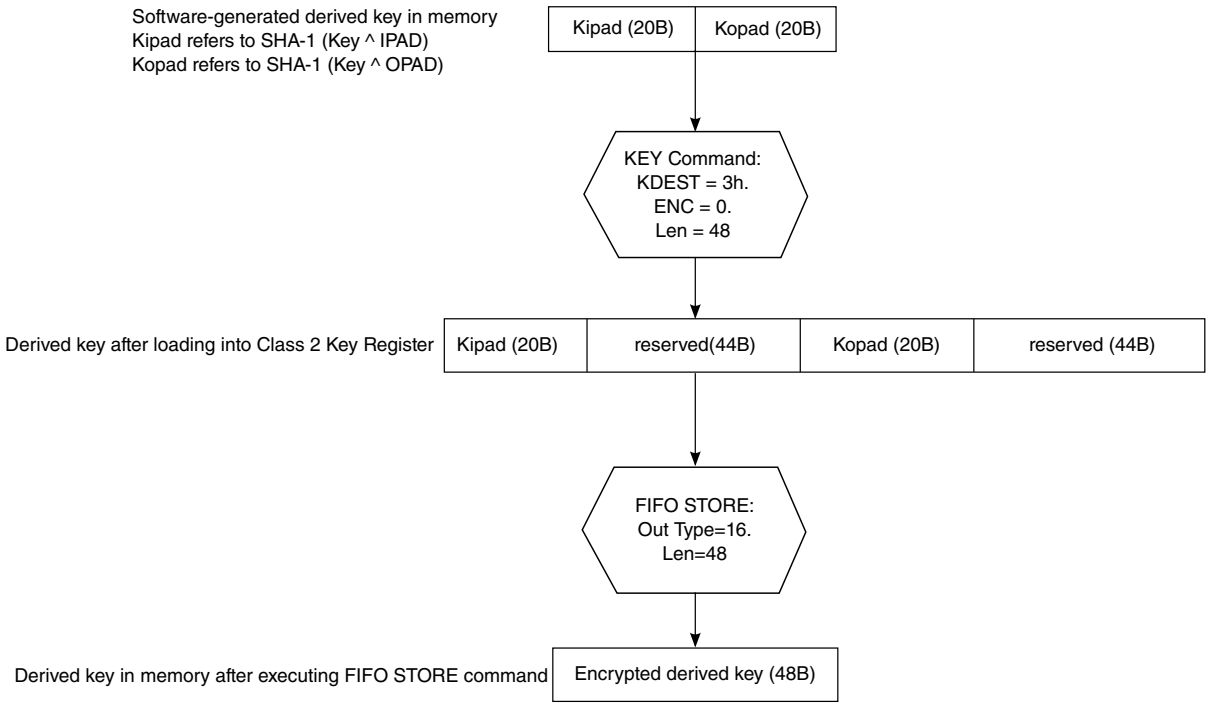


Figure 10-16. Derived keys in memory and in the Class 2 Key Register

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.

10.10.5.2.3 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

10.10.5.3 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.

10.10.5.4 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

10.10.5.5 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.

10.10.6 AES accelerator (AESA) functionality

The Advanced Encryption Standard Accelerator (AESA) module is a hardware co-processor capable of accelerating the advanced encryption standard (AES) cryptographic algorithm. Note that the AESA implements GCM mode. AESA is invoked by setting ALG=10h in the ALGORITHM OPERATION command.

10.10.6.1 Differences between the AES encrypt and decrypt keys

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, the two forms are considered the same key because one can be derived from the other.) The decrypt form of the key is different from the encrypt form of the same 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. If CAAM is told to do a decrypt operation but the key register currently contains the encrypt form of the key, CAAM first goes through the steps required to derive the decrypt key from the encrypt key, and then performs the decryption operation. To increase the performance of

decryption operations, the AESA CHA can be told to start with the decrypt form of the key. If the OPERATION command specifies the DK (Decrypt Key) bit as 1, the AESA CHA will assume that the key register already contains the decrypt form of the key, and will skip the steps to derive the decrypt key from the encrypt key. But if DK = 0 when a decryption command is executed, the AESA CHA will perform the steps needed to derive the decrypt form of the key.

If an AES encryption operation is performed and then within the same job the AES key is stored as a black key, the black key will be the encrypt form of the AES key. But if the AES key is stored as a black key after an AES decryption operation is performed, the encrypted key will be the **decrypt** form of the AES key. Software must keep track of whether the black key is the encrypt form or the decrypt form of the AES key. Failure to use the correct key form in a decryption operation will yield incorrect results. If a decrypt form key was loaded in the key register but DK = 0, the key would be modified as if it were an encrypt key, and consequently, the wrong key value would be used in the decryption operation. If an encrypt form key was loaded in the key register but DK = 1, CAAM would use the encrypt key assuming that it was a decrypt key, and again the wrong key value would be used in the decryption operation.

The difference between the encrypt key and the decrypt key must be taken into account when sharing an AES key between jobs. When an AES key is shared from an encryption job it is the encrypt form of the key that is shared. But when a subsequent AES decryption job shares the key from a previous decryption job, the key that is shared is a **decrypt**key. If the key that is shared is a decrypt key the DK bit should be set to 1, which tells CAAM to skip the decrypt key derivation steps. Note that a subsequent Shared Descriptor will receive a **decrypt** key in the key register if sharing occurred (i.e. the shared-to job started while the shared-from job was still in a DECO), but may receive an **encrypt** key in the key register if sharing did not occur and the key had to be reloaded from memory. A JUMP command with TEST CONDITION set to SHRD (see Table 10-88) can be used to determine whether the OPERATION command should be executed with DK = 0 or DK = 1.

10.10.6.2 AESA modes of operation

The following modes are supported by AESA:

- Electronic codebook (ECB)
- Cipher block chaining (CBC)
- CBC mode with cipher text stealing (CBC-CS2)
- 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)
- Galois/Counter mode (GCM)

AES modes can be classified into these categories:

- Confidentiality (ECB, CBC, CBC-CS2, CTR, OFB, CFB128)
- Authenticated Confidentiality (CCM, GCM)
- 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.

10.10.6.3 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 AESA 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.
- KEY SIZE, MODE and DATA SIZE can be written in any order. The operation will begin after all of these have been written.Also, for all AES modes, the bit offset in the Data Size Register must be zero. Failure to comply with these requirements will generate an error in the CCB Status Register.
- If ICV-only jobs are created (no data or no additional data to be processed, only ICV to be checked) in modes that support ICV check, the ICV_TEST mode bit should be set, and the AS mode field should be programmed to Update.
- The default register updates performed by the DECO/CCB complex when an AES CHA is utilized via the register-based service interface are described in chapter DECO/CCB behavior for jobs started via the register service interface. Similarly, the updates performed for jobs originating from other service interfaces or when sharing occurs are described in chapter DECO/CCB default actions for one-off jobs and DECO/CCB actions when sharing descriptors, respectively.

10.10.6.4 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.

10.10.6.5 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.

10.10.6.5.1 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 AES by setting it to 10h .

10.10.6.5.2 AES ECB mode use of the Context Register

The AES ECB mode does not use the Context Registers except when the 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 10-214. Context usage in ECB mode

Context DWord	Definition	
	ECB	ECB with ICV/TEST = 1
0	-	ERROR CODE
1	-	

10.10.6.5.3 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.

10.10.6.5.4 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.

10.10.6.5.5 AES ECB Mode use of the Key Size Register

The number of bytes in the ECB key must be written to the Key Size register. The KEY SIZE, MODE and DATA SIZE can be written in any order. Processing starts after all of them have been written. Any value other than 16, 24, or 32 causes the key-size error to be generated.

10.10.6.6 AES CBC, CBC-CS2, OFB, CFB128 modes

The CBC, CBC-CS2, OFB, CFB128 modes are considered together because of their similarities and are described in this table.

Table 10-215. AES CBC, CBC-CS2, OFB, CFB128 modes

Name	Abbreviation	Function
Cipher-Block Chaining mode	CBC	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 **NOTE:** CBC mode uses both forward and inverse AES cipher. OFB and CFB use only forward AES cipher.
Cipher-Block Chaining mode with CipherText Stealing (CTS)	CBC-CS2	CipherText Stealing is a variant of CBC mode for use when the plaintext is not a multiple of the block size. CAAM supports Ciphertext Stealing variant CS2 as defined in the NIST Addendum to Special Publication 800-38A: "Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode".
Cipher FeedBack mode (128-bit)	CFB128	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.
Output FeedBack mode	OFB	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.

NOTE

References to CBC mode in the following text also apply to the CBC-CS2 mode.

10.10.6.6.1 AES CBC, OFB, and CFB128 modes use of the Mode Register

The AES CBC, OFB, and CFB128 modes use the Mode Register as follows:

- 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, CBC-CS2, OFB, and CFB128, these values are 10h, 12h, 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 AES by setting it to 10h .

10.10.6.6.2 AES CBC, OFB, and CFB128 modes use of the Context Register

The AES CBC, OFB, and CFB128 modes use the Context Register as follows:

- AES 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 10-216. Context usage in CBC, OFB, CFB modes

Context DWord	Definition
0	IV [127:64]
1	IV [63:0]

10.10.6.6.3 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. This register 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 (but not CBC-CS2) and CFB128 modes, otherwise the data-size error is generated.
- Only OFB and CBC-CS2 decrement the value in this register with every processed block.

10.10.6.6.4 AES CBC, OFB, and CFB128 modes use of the Key Register

The AES CBC, OFB, and CFB128 modes uses 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.

10.10.6.6.5 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.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

10.10.6.7 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.

10.10.6.7.1 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 AES by setting it to 10h .

10.10.6.7.2 AES CTR mode use of the Context Register

The AES CTR mode uses the Context Register as follows:

- CTR uses context dwords 2 and 3 to provide initial counter value (CTR0). This value is incremented 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 dwords 2 and 3 in place of CTR0 and with all-zero block provided as input data through the FIFO without overwriting the context.

Table 10-217. Context usage in CTR mode

Context dword	Initial-input definition	Context-switching definition
0	-	-
1	-	-
2	CTR0 [127:64]	CTRi [127:64]
3	CTR0 [63:0]	CTRi [63:0]

10.10.6.7.3 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.

10.10.6.7.4 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.

10.10.6.7.5 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. Any value other than 16, 24, or 32 will cause Key Size error to be generated.

10.10.6.8 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.

10.10.6.8.1 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_TEST 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 10-218. 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_TEST 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 10-219. 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_TEST:w 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_TEST 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 AES by setting it to 10h .

10.10.6.8.2 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 dwords of the context contain the MAC value.

Table 10-220. Context usage in XCBC-MAC and CMAC modes

Mode	Context dword	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]	-

10.10.6.8.3 AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register

The AES XCBC-MAC and CMAC modes use the ICV Size Register as follows:

- This 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 to 16 bytes. If this register is 0, the size of ICV is 16 bytes.

10.10.6.8.4 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.

10.10.6.8.5 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.

10.10.6.8.6 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.
- 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.

10.10.6.8.7 ICV checking in AES XCBC-MAC and CMAC modes

Automatic ICV checking is enabled by setting the ICV_TEST 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.

10.10.6.9 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.

10.10.6.9.1 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.

10.10.6.9.2 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.

10.10.6.9.3 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_TEST 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_TEST 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 10-221. 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_TEST 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_TEST 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. CCM uses the key in the Class 1 Key Register.Setting the DK bit causes the illegal-mode error.
- The Algorithm (ALG) field is used to activate AES by setting it to 10h .

10.10.6.9.4 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 dword 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 dword 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 dwords 4 and 5. The final computed MAC is written (in clear) to dwords 0 and 1.

Table 10-222. Context usage in CCM mode encryption

Context DWord	Initial-input definition	Intermediate definition	Final-output definition
0	B0[127:64]	intermediate MAC state	MAC[127:64]
1	B0[63:0]	intermediate MAC state	MAC[63:0]
2	CTR0[127:64]	CTR[127:64]	-
3	CTR0[63:0]	CTR[63:0]	-
4	-	E(CTR0)[127:64], [1]	E(MAC)[127:64]
5	-	E(CTR0)[63:0[1]	E(MAC)[63:0]
6	-	AAD size, MAC size; see Table 10-224	-

1. E(x) means encrypted x

Table 10-223. Context usage in CCM mode decryption

Context DWord	Initial-input definition	Context-switching Definition	Final-result definition
0	B0[127:64]	intermediate MAC state	MAC[127:64]

Table continues on the next page...

Table 10-223. Context usage in CCM mode decryption (continued)

Context DWord	Initial-input definition	Context-switching Definition	Final-result definition
1	B0[63:0]	intermediate MAC state	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 10-224. 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

10.10.6.9.5 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.

10.10.6.9.6 AES CCM mode use of the Key Register

CCM key must be written to this register; it is always an encryption key.

10.10.6.9.7 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.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

10.10.6.9.8 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_TEST 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.

10.10.6.10 AES GCM mode

AES GCM mode provides the following:

- Data confidentiality using counter mode (CTR). 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", but it is the user's responsibility to ensure that the same key value is not used again following a reset.
- Authentication (assurance of integrity) of the confidential data using a universal hash function (GHASH) that is defined over a binary Galois (that is, finite) field. GCM can also provide authentication assurance for additional data (AAD) that is not encrypted.
- Stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both of the following:
  - Accidental modifications of the data
  - Intentional, unauthorized modifications

10.10.6.10.1 GMAC

If the GCM input is restricted to data that is not encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data.

10.10.6.10.2 GCM data types

In the GCM mode, the FIFO data type must be set to the message data type for textdata (payload), AAD type for additional data, IV type for IV data and ICV type for the received ICV. These data types must always be provided in the following order:

1. IV
2. AAD
3. Message data

Any of these may be missing.

10.10.6.10.3 IV processing

IV is processed using GHASH function if the size of IV is not 12 bytes. The result of IV processing is the initial counter (Y0) value used for encryption/decryption. GHASH function is also performed on AAD and textdata before the MAC can be computed.

10.10.6.10.4 GCM initialization

GCM initialization is completed when all of the IV data is processed and the initial counter value (Y0) is computed as a result. For that to happen, IV data needs to be supplied through the Input Data FIFO and the FIFO data type must be set to IV.

10.10.6.10.5 AES GCM mode use of the Mode Register

The AES GCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption. Even though operations performed in either case are identical, the authentication is done of the cipher text in parallel with decryption when ENC = 0, and after encryption of each block when ENC = 1.
- The ICV_TEST bit must be set for GCM to compare computed MAC 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, GCM does not expect received ICV to be supplied after textdata. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- The Algorithm State (AS) field is defined for GCM as shown in this table:

Table 10-225. Mode Register[AS] operation selections in AES GCM

Operation	Value	Description
INITIALIZE	1h	Message is processed in multiple sessions and the current session processes final part of IV or textdata; do the final GHASH step, but do not compute MAC.

Table continues on the next page...

Table 10-225. Mode Register[AS] operation selections in AES GCM (continued)

Operation	Value	Description
		NOTE: This AS state does not indicate initialization in GCM; instead, it means that the final step of the GHASH function is to be performed. In general, whenever the final GHASH iteration needs to be computed (either for GHASH(IV) or GHASH(AAD, ciphertext)), and the current message size provided in the Data Size Register is not equal to the total size for either IV, AAD, or textdata, AS should be set to INITIALIZE (1h). Consequently, an AS = 1h also indicates that the Context Registers 6-7 need to provide the total length of IV, AAD, or textdata for this to be accomplished.
INITIALIZE/ FINALIZE	3h	Message is processed in multiple sessions and the current session is the last. The final MAC is computed.
UPDATE	0h	Message is processed in multiple sessions (descriptors) and the current session is not the last. The descriptor contains a non-final part of IV, AAD, textdata (IV, AAD or textdata split between descriptors). If decryption is requested, and data size is not written or is set to 0, and ICV_TEST 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	2h	Message is processed in a single session. MAC is computed.

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV_TEST bit is set to 1, the Illegal Mode error will be generated except for CICV-only jobs.

Proper AS field settings

Assume that a message has IV, AAD, and textdata and each of these types is split into two sessions (descriptors). The first IV descriptor should have AS set to "Update", the second IV Descriptor should have AS set to "Initialize", both AAD Descriptors and the first textdata descriptor should have AS field set to "Update", and the final Descriptor sets AS to "Initialize/Finalize".

- The Additional Algorithm Information (AAI) field must be set to 90h for GCM to be activated. GCM uses the key in the Class 1 Key Register.Setting the DK bit causes an illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

10.10.6.10.6 AES GCM mode use of the Context Register

The AES GCM mode uses the Context Register as follows:

- New message processing does not need any data provided in the context. All of the context data is written back by the GCM mode and needs to be restored before the next data chunk is to be processed in the multi-session processing. The final MAC is written in the context dwords 0-1.

- The initial counter value required for encryption/decryption is derived from IV and written to dwords 4-5. It is also required for the MAC computation.
- The incremented counter is placed in dwords 2-3 and is updated with every encrypted/decrypted block.
- Bit sizes of IV, AAD and textdata are required for GHASH computation and are accumulated in dwords 6-7 when multi-session processing is used.

Table 10-226. Context usage in GCM mode

Context DWord	Context-switching definition	Final-result definition
0	MAC[0:63]	MAC[0:63]
1	MAC[64:127]	MAC[64:127]
2	Yi[0:63]	-
3	Yi[64:127]	-
4	Y0[0:63]	-
5	Y0[64:127]	-
6	IV bit size (during GHASH of IV), AAD bit size (during message processing)	-
7	textdata bit size	-

10.10.6.10.7 AES GCM Mode use of the Data Size Register

The AES GCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed (including IV, AAD and textdata) must be written to the Data Size register (IV and AAD sizes must include padding to the 16 byte boundary).
- he 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.
- GCM decrements the value in this register with every processed block.
- Message splitting must be done only on a 16-byte boundary.

10.10.6.10.8 AES GCM mode use of the Class 1 IV Size Register

The Class 1 IV Size register is written with the number of bytes in the last IV block. If the total IV size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the IV used in the GHASH computation. To do this, GCM also uses the fact that IV size padded to a 16-byte boundary is written to the Data Size register.

10.10.6.10.9 AES GCM mode use of the AAD Size Register

The AAD Size register is written with the number of bytes in the last AAD block. If the total AAD size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the AAD used in the GHASH computation. To do this, GCM also uses the fact that AAD size padded to a 16-byte boundary is written to the Data Size register.

10.10.6.10.10 AES GCM mode use of the Class 1 ICV Size Register

The AES GCM mode uses the Class 1 ICV Size Register as follows:

- This Class 1 register is used to provide ICV/MAC byte-size when it is other than 16 bytes. In that case, the remaining bytes of the ICV/MAC written to the context is zero.
- If the ICV mode bit is set, the Class 1 ICV Size register also determines the number of bytes in the received ICV. Supported values for ICV size are 4 to 16 bytes. If this register is 0, ICV size will be 16 bytes.

10.10.6.10.11 AES GCM mode use of the Key Register

GCM key must be written to this register; it is always an encryption key.

10.10.6.10.12 AES GCM mode use of the Key Size Register

The AES GCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Key Size register.
- Any value other than 16, 24, or 32 causes key-size error to be generated.

10.10.6.10.13 AES GCM mode use of the ICV check

The AES GCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting ICV_TEST 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 jobs. 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. 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 GCM written to the Class 1 ICV Size register.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

10.11 Trust Architecture modules

The CAAM Trust Architecture functions may be performed in the run-time integrity checker (RTIC), Secure Memory and the secure key module. Other Trust Architecture Modules are special features for protecting data or keys.

10.11.1 Run-time Integrity Checker (RTIC)

The run-time integrity checker (RTIC) is a component of CAAM that is used to ensure the integrity of the peripheral memory contents and assist with boot authentication. The RTIC has the ability to verify the memory contents during system boot and during run-time execution. If the memory contents at runtime fail to match a reference hash signature, then a security violation is asserted. This security violation should then be captured by a monitoring device on the platform.

10.11.1.1 RTIC modes of operation

The RTIC modes of operation are described in this table.

Table 10-227. RTIC modes of operation

Mode	Description
One-time hash mode	• Used during high assurance boot for code authentication or one time integrity checking • Generates and stores a reference hash result internally and signals an interrupt to the processor
Continuous hash mode	• Used at run time to continuously verify the integrity of memory contents • Checks a re-generated hash against an internally stored reference value and interrupts the processor only if an error occurs

10.11.1.2 RTIC initialization and operation

RTIC supports integrity checking of up to four, independent memory blocks. RTIC's Hash Register File stores a reference hash for each memory block. During the boot stage integrity checking, each independent memory block's content is hashed and the result is stored in the hash register file. At boot-time, the memory contents are read and hashed (authenticated) as quickly as possible (RTIC Throttle Register should be set to 00h) to minimize the performance impact at startup. The reference hash result for each memory block is stored in RTIC for the processor to compare against the signed code hash value. Once RTIC has finished hashing the memory block, RTIC interrupts software, which can

then check the generated hash value against digitally-signed hash value(s) stored with the code. Software policy determines what actions to take in the event of a hash mismatch at boot time. Note that in chips supporting high-assurance boot, RTIC's boot-image hashing may take place after secure-boot software validates the first code to execute. This means that any unauthorized code modification would either be caught by the secure boot software before RTIC runs, or trusted software would detect the hash mismatch after RTIC had integrity-checked the boot image.

After trusted boot software has verified the boot image, the software can put RTIC into run-time mode to ensure that the boot image remains uncorrupted. In run-time mode, RTIC periodically reads a small section of memory, waits for a specified period of time, and then reads another small section of memory. During this process, RTIC computes a hash of the software image. When RTIC has eventually read the entire software image, it compares the newly computed hash with the reference hash that was validated earlier. If the RTIC hardware detects a hash mismatch, RTIC generates an interrupt and signals a security violation to the chip's security monitor (SecMon) hardware. If the hash matches, RTIC starts over and re-validates the software image. This process repeats until the chip is powered down or RTIC checking is turned off.

10.11.1.3 RTIC use of the Throttle Register

The RTIC scan rate is controlled using the Throttle Register. This allows the user to trade off the software image revalidation rate against memory bandwidth utilization. Depending on the settings, the software image might be revalidated every few seconds or every few days. RTIC also implements a watchdog timer that can be used to ensure that an attacker isn't able to block RTIC's access to memory for an extended period of time. If a DMA read error, illegal address/length error, RTIC Watchdog time-out, or hash mismatch occurs, the RTIC enters an error state and signals a security violation. A hardware reset is required to resume operation.

10.11.1.4 RTIC use of command, configuration, and status registers

The RTIC controller holds the command/configuration registers, which are programmed through CAAM's register interface. RTIC uses CAAM's DMA interface only to read memory contents. The command/control registers are used to:

- Set the DMA and throttle level
- Specify which memory blocks to hash (one time or continuously)
- Enable/Disable/Clear interrupts
- Enable one-time or run-time hashing, software reset, and clear interrupts

A status register in the RTIC indicates the current state of the controller, which includes:

- Interrupt status
- Processing status
- Error status

The controller also contains a comparator to check the generated hash value against the reference hash value.

10.11.1.5 Initializing RTIC

At boot time, RTIC can be used to accelerate software-image verification. This is accomplished by first selecting the hash algorithm (SHA-256 or SHA-512) and one or more RTIC memory blocks by writing to the RTIC Control Register, specifying the areas of memory to be hashed by writing one or more pairs of RTIC Memory Block Address and Length registers or creating a scatter/gather table and, if necessary, altering the endianness settings via the RTIC Control Register ,and then writing to the RTIC Command Register to initiate the hashing operations.

At chip-initialization time, RTIC is configured for run-time mode operation by writing to the RTIC Control Register, the RTIC Throttle Register the RTIC Watchdog Register and the RTIC Memory Block Address and Length registers and, if necessary, writing to the RTIC Control Register to change the endianness settings, and then writing to the RTIC Command Register to put RTIC into run-time mode.

10.11.1.6 RTIC Memory Block Address/Length Registers

Up to four independent memory blocks can be hashed by the RTIC, each with its own message digest (reference hash value). The RTIC scans through the memory blocks in the order they are defined in the RTIC Memory Block Address registers. Each of the four Memory Blocks can be divided into two separate segments, with separate starting addresses and segment lengths. RTIC computes the hash over each Memory Block that is enabled by first reading segment 0 of the Memory Block and then appending segment 1.

Since there can be two segments per Memory Block, each memory block (A, B, C, D) is defined by two address/length register pairs (RTIC Memory Block Address 0 Register / RTIC Memory Block Length 0 Register and RTIC Memory Block Address 1 Register / RTIC Memory Block Length 1 Register). For each memory block, starting at the address indicated in the RTIC Memory Block Address 0 Register RTIC reads the number of memory bytes specified in the RTIC Memory Block Length 0 Register. When that is complete, RTIC starts at the address indicated in the RTIC Memory Block Address 1 Register and reads the number of memory bytes specified in the RTIC Memory Block

Length 1 Register. If a Length Register is set to zero, RTIC skips over that memory segment. Once the specified number of bytes are read from both segments within the memory block, the data is hashed and stored (Hash-Once mode) or compared to the reference value (Run-Time mode). Information on additional registers used for RTIC configuration can be found in the sections describing the RTIC registers in CAAM register block 6.

10.11.2 CAAM virtualization and security domain identifiers (SDIDs)

This section describes the CAAM features that are intended to support virtualization of the CAAM hardware; that is, the ability to share the CAAM functionality among multiple software entities.

10.11.2.1 Access Control

The hardware resources of CAAM can be shared among multiple users (processors or processes). Each of these users can submit crypto jobs to CAAM or can access portions of Secure Memory. Many of the different CAAM resources are accessible through separate register pages, which are spaced within the address space to match the size of memory pages. This is intended to allow access to different resources to be controlled via CPU MMUs (for processes) or System MMUs (for processors). Processor access to CAAM resources can also be controlled by setting CAAM's internal access control hardware. This hardware controls access to individual register pages using certain bus signals (DIDs). The DID value that grants access to a particular CAAM register page is configured by writing into registers within CAAM register page 0. (See Job Ring a DID Register - most significant half (JR0DID_MS - JR2DID_MS), RTIC DID Register for Block a (RTICA_DID - RTICD_DID).) A CAAM register page is accessible if the DID associated with the register read or write matches the DID in the appropriate register within CAAM register page 0.

CAAM also controls access to register pages based on the TrustZone bus signal. This signal is usually called "nonsecure" or "ns". A 1 on the ns signal indicates a NonSecureWorld bus transaction, and a 0 on the ns signal indicates a SecureWorld transaction. SecureWorld is a higher-privileged mode than NonSecureWorld, so bus transactions with ns=0 are higher privileged than ns=1 transactions. A CAAM register page can be made accessible only to TrustZone SecureWorld by setting the page's access control register appropriately. If a CAAM register page is configured for access by the

TrustZone NonSecureWorld associated with a particular DID value, it can also be accessed via a SecureWorld bus transaction (ns=0) with that particular DID value, but not by SecureWorld or NonSecureWorld bus transactions with some other DID value.

Note that a Job Ring owner's DID value forms part of the SDID value associated with that Job Ring. The SDID value is used to cryptographically separate Black Keys, Blobs and Trusted Descriptors. A CAAM descriptor executed from a particular Job Ring can access only Black Keys, Blobs and Trusted Descriptors associated with that Job Ring's SDID value. Different Job Rings can be configured with different SDID values in order to prevent jobs executing in one Job Ring from accessing Black Keys, Blobs or Trusted Descriptors belonging to other Job Rings.

10.11.2.2 Virtualization

CAAM has been designed so that it can be "virtualized", that is, it can be shared among multiple software entities while still maintaining individual security protections for each of these entities. These software entities might include guest operating systems running under a hypervisor that allocates the chip's hardware resources among the guest OSs.

10.11.2.3 Security domain identifiers (SDIDs)

CAAM implements 4096 security domain identifier (SDID) values that system control software (for example: hypervisor, kernel, operating system) can associate with different software entities. SDIDs are used to provide data separation between software entities, and are used with

- black keys
- blobs
- trusted descriptors

The least significant 5 bits of the SDID (the TZ and DID bits) are also used with Secure Memory. An SDID is a static value that must be maintained across power cycles as it is used to provide data separation even across power cycles. A unique SDID value can be associated with each software entity, or the same SDID value can be associated with multiple software entities, or multiple SDID values can be associated with a single software entity. The SDID values must be assigned at boot configuration time, and the same SDID must always assigned to the same software entity. In the case of a guest OS running under a hypervisor, the guest OS may assign some of its different SDID values to processes under the guest OS's control. Note that the hypervisor is itself one of the software entities that can utilize CAAM's functionality, and can assign itself as many SDID values as it wishes. CAAM imposes no restrictions on how these SDID values are assigned, but simply uses the SDID values to control how data is shared among SDID

assignees, or kept private to each assignee. SDID values are assigned to RTICand to Job Rings by writing to registers in page 0 of CAAM's register address space. The sections below describe how CAAM uses the SDID values to "virtualize"

- Secure Memory partitions
- black keys
- blobs
- trusted descriptors

10.11.2.4 TrustZone SecureWorld

CAAM recognizes TrustZone SecureWorld as a unique software entity with special privileges, and identifies SecureWorld using a special "TZ" security identifier. All CAAM registers that are used to hold 12-bit SDID values also have a separate TZ bit. Hardware signals ensure that only TrustZone SecureWorld can write to TZ bits. TrustZone SecureWorld indicates that specific CAAM resources belong to SecureWorld by setting the TZ bit to 1. SDIDs are used to provide data separation between software entities, and are used with

- black keys
- blobs
- trusted descriptors

This allows SecureWorld

- to claim CAAM Job Rings for its exclusive use
- to generate black keys that cannot be encrypted or decrypted by non-SecureWorld
- to encapsulate and decapsulate blobs that cannot be encapsulated or decapsulated by non-SecureWorld
- to create trusted descriptors that can be executed in any Job Ring
- to claim CAAM Secure Memory partitions for its exclusive use

The sections below describe how CAAM enforces these SecureWorld privileges.

10.11.3 Special-purpose cryptographic keys

CAAM provides protection of session keys by means of black keys,integrity protection of CAAM descriptors by means of trusted descriptors, and protection of long-term secrets by means of blobs. All of these protection mechanisms make use of special-purpose cryptographic keys managed by CAAM.

10.11.3.1 Initializing and clearing black and trusted descriptor keys

The CAAM hardware implements special black key key encryption keys (see Black keys) and trusted descriptor signing keys (see Trusted descriptors). These must be initialized to random values each time that CAAM powers up. These keys are cleared when CAAM is in the fail mode. The hardware implementation ensures that only CAAM itself 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).

10.11.3.2 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 hardware 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 hardware 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.

10.11.3.3 Trusted descriptors and TDSK

The CAAM hardware 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.

10.11.3.4 Master key and blobs

The special cryptographic key used for blobs is the 256-bit master key. The CAAM hardware 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.

10.11.4 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, modified via the appropriate TZ/SDID value. Thus, each security domain (and TrustZone SecureWorld) has its own private black keys, which cannot be decrypted by the user of a different security domain identifier. 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.

10.11.4.1 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. CAAM does not mix the length of the encrypted key into the value of the key encryption key when using the high assurance black key scheme, because the MAC-tag prevents misrepresenting the length of the encrypted key. In AES-CCM encryption the AES algorithm is always used in the "encryption" direction regardless of whether the key is being encrypted or decrypted, so in the high-assurance black key scheme encapulation and decapsulation require approximately the same amount of time.

10.11.4.2 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.

10.11.4.3 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 10-36, values 10h-27h).

10.11.4.4 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).

10.11.4.5 Avoiding errors when loading red and black keys

There are many ways to load red and black keys. Some of these ways will be successful, and other ways will not be successful. In many cases an unsuccessful key load will generate an error message, but there are a few cases that CAAM cannot detect, so these do not produce error messages. In those cases special care should be taken.

Key Type	Error	Error Indication	Comments
Black Key	Black Key loaded as Red Key (Black Key loaded into a key register using a LOAD command or a KEY command with ENC = 0)	The LOAD or KEY operation will complete normally, but since the key register will contain a Black Key rather than a Red Key, the key value will be incorrect. If the key is then used in an encryption or decryption operation, the operation will complete normally, but incorrect results will be produced.	Since all possible values are permissible in a Red Key, CAAM cannot distinguish between a Black Key and a Red Key. Software should take care to distinguish Red Keys from Black Keys either by location or via software tags.

Table continues on the next page...

Key Type	Error	Error Indication	Comments
Red Key or ECB-encrypted Black Key	non-ECB-Encrypted Key loaded as an ECB-Encrypted Black Key (Key loaded into a key register using a KEY command with ENC = 1 and EKT = 0)	The KEY command will decrypt the key as if it were an ECB-encrypted Black Key. The Key command will complete normally, but since the key register will contain a Black Key rather than a Red Key, the key value will be incorrect. if the key is then used in an encryption or decryption operation, the operation will complete normally, but incorrect results will be produced.	Since all possible values are permissible in an ECB-encrypted Black Key, CAAM cannot distinguish between a CCM-encrypted Black Key or a Red Key and an ECB-encrypted Black Key. Software should take care to distinguish Red Keys and CCM-encrypted Black Keys from ECB-encrypted Black Keys either by location or via software tags.
CCM-encrypted Black Key	CCM-encrypted Black Key loaded as CCM-encrypted Black Key, but key value is corrupted (Key loaded into a key register using a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	The MAC check failed when the CCM-encrypted Black Key was decrypted.
Red Key or ECB-encrypted Black Key	Non-CCM-encrypted Black Key loaded as CCM-encrypted Black Key (Key loaded into a key register using a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	When CAAM tried to verify the MAC tag, the check failed because the key did not have a MAC tag.
CCM-encrypted Black Key	CCM-encrypted Black Key loaded by the wrong security domain (Key loaded into a key register using a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	When CAAM tried to verify the MAC tag, the check failed because the wrong SDID value was used in deriving the key encryption key.
Any Key	Specified size too large for key register Value in KEY command's LENGTH field is larger than the Class 1 Key Register or Class 2 Key Register (depending upon the destination specified in the Key command).	Key Size Error Job termination status word: Source = 2h (CCB); ERRID = 3h (Key size error)	LENGTH is less than 16 when reading a key from a Secure Memory key partitionSince all possible values are permissible in an ECB-encrypted Black Key, CAAM cannot distinguish between a CCM-encrypted Black Key or a Red Key and an ECB-encrypted Black Key. Software should take care to distinguish Red Keys and CCM-encrypted Black Keys from ECB-encrypted Black Keys either by location or via software tags.
Any Key	Specified size too small when loading key from Secure Memory key partition Value in KEY command's LENGTH field is less than 16.	Key Size Error Job termination status word: Source = 2h (CCB); ERRID = 3h (Key size error)	The length of the key must be at least 16 bytes when loading a key from a Secure Memory key partition. This restriction is intended to prevent the key value from being determined by exhaustively testing subsets of the actual key.

10.11.4.6 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.

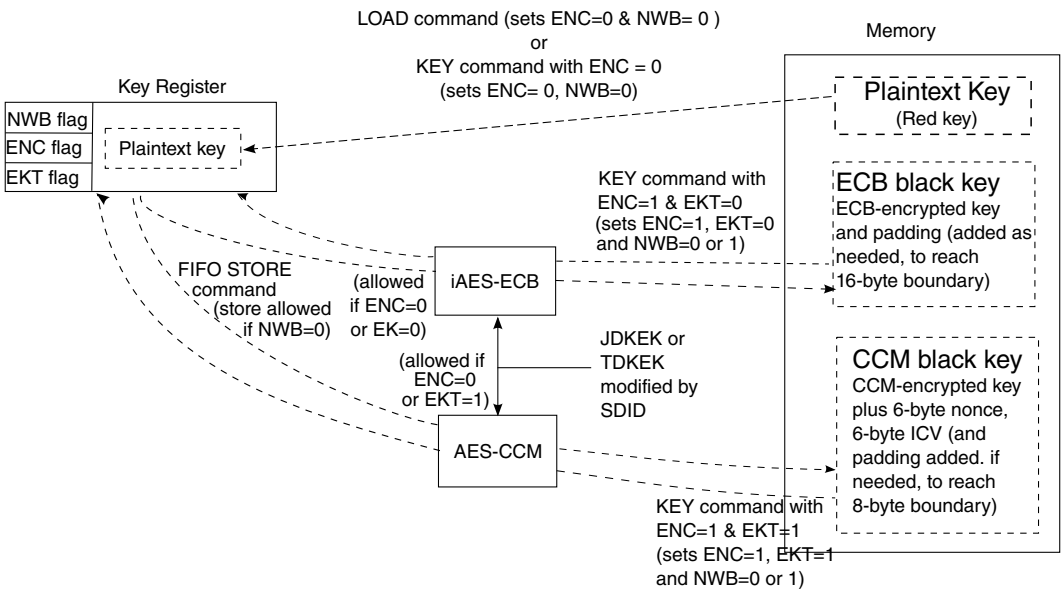


Figure 10-17. Encapsulating and decapsulating CAAM black keys

The cryptographic key used to encrypt or decrypt black keys is held in the 256-bit JDKEKR or TDKEKR, and this key's value is modified via the appropriate TZ/SDID value before being used in a black key operation on behalf of some descriptor. The TZ/SDID value is taken from the job ring SDID register, depending upon where the descriptor is executed. Job 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 so they cannot be decrypted by Job Descriptors.

The black keys used by each SDID value are encrypted using a different modification of the JDKEK or TDKEK in order to provide cryptographic separation between the keys used in different security domains. Note that TrustZone trusted descriptors always use TZ = 1b, SDID = 000h for the JDKEK/TDKEK modification, regardless of the Job Ring in which the descriptor is executed.

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.

10.11.4.7 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 10-228. 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.)

10.11.4.8 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.

10.11.5 Trusted descriptors

Trusted descriptors provide a means for trustworthy software to create trusted "applets" that can be safely executed by less trustworthy software.

10.11.5.1 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.

10.11.5.2 Trusted-descriptor key types and uses

When CAAM is in trusted mode or secure mode, the hardware (Special-purpose cryptographic keys) 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.

10.11.5.3 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.

10.11.5.4 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.

10.11.5.5 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.

10.11.5.6 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 JRaDID register. Proper configuration is required to ensure that only trusted software can write to any JRaDID 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 LAMTD bit) by trusted boot software.
- The system uses the operating system or hypervisor to control access to the address block that includes the JRaDID 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 JRaDID 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 register address block containing a particular Job Ring's control registers, and denying access to that block to other processes.

10.11.5.7 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.

10.11.5.7.1 Trusted descriptors and descriptor-header bits

Descriptor headers contain a 2-bit field related to trusted descriptors. See HEADER command for a full explanation of the descriptor header.

The 00 value in the TDES field indicates an ordinary job descriptor. The 11 value indicates a candidate trusted descriptor, that is, a descriptor that CAAM should convert into a trusted descriptor by affixing a signature. When CAAM executes a candidate trusted descriptor, it checks to see if the AMTD (allow make trusted descriptor) bit is set in the Job Ring's JRaDID register. If not, the candidate trusted descriptor is not converted to a trusted descriptor and the job terminates with an error. If AMTD=1, CAAM changes the TDES field value to 10 if the candidate trusted descriptor is being created in a Job Ring owned by TrustZone nonSecureWorld, but changes the TDES field to 01 if the Job Ring is owned by TrustZone SecureWorld. CAAM then either affixes a signature to the new trusted descriptor, or executes the trusted descriptor, or both, depending upon the option in the SIGNATURE command at the end of the descriptor.

10.11.5.7.2 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.
- A TrustZone non-SecureWorld trusted descriptor can be executed only within a Job Ring that has the same SDID value as the Job Ring in which the trusted descriptor

was created. The reason for this restriction is that Job Rings may be owned by different security domains that do not trust each other's TrustZone non-SecureWorld trusted descriptors. This restriction is enforced by including the SDID of the Job Ring's JRaDID register in the signature computation, both when creating the trusted descriptor and before executing the trusted descriptor.

- A TrustZone SecureWorld trusted descriptor can be executed within any Job Ring, regardless of that Job Ring's SDID or TZ value. TrustZone SecureWorld trusted descriptors can be created only in a Job Ring owned by TrustZone SecureWorld. This allows TrustZone SecureWorld to create special trusted descriptors that are trusted by all security domains. The TDES field in the trusted descriptor's HEADER command is used to distinguish SecureWorld trusted descriptors from non-SecureWorld trusted descriptors.
- 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), In-Line Descriptor (see INL field in Table 10-95), or Replacement Job Descriptor (see RJD field in Table 10-95).
- 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.

10.11.6 Blobs

CAAM can protect data in a cryptographic data structure called a blob, which provides both confidentiality and integrity protection.

10.11.6.1 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 master secret key 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 10-57 shows the format of the PROTINFO field for the blob protocol, and Table 10-58 describes the bit values.

10.11.6.2 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 using a secure non-volatile key. [10] Using a unique non-volatile key for each device prevents data encrypted on one device from being copied and decrypted on a different device, which might compromise the secrecy of the data.

10.11.6.3 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).

- FIPS PUB 197, *Advanced Encryption Standard (AES)*, November 26, 2001.
- FIPS PUB 180-2, SECURE HASH STANDARD, August 1, 2002.
- SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012. Draft SP800-90B, *Recommendation of the Entropy Sources Used for Random Bit Generation*, August 2012.
- SP800-38C, *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, May 2004.

10. The data is actually encrypted with a randomly generated blob key, and it is that blob key that is encrypted using the secure non-volatile key

- SP800-56A, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, March 2007.
- SP800-56C revision 1, *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*, April 2018.
- 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. 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 is specified in SP800-90A, using the Hash_DRBG with SHA-256 as the hash function. It gets entropy from a live entropy source intended to comply with SP800-90B. The random number generator has a security strength of 256 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 SP800-56C (sec. 5.8.1), using SHA-256 as the hash function.

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.

10.11.6.4 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

10.11.6.5 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, that is, 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 10-229. 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) • Encryption modes: AES-ECB and AES-CCM • Encryption keys: JDKEK and TDKEK	
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	

10.11.6.5.1 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.

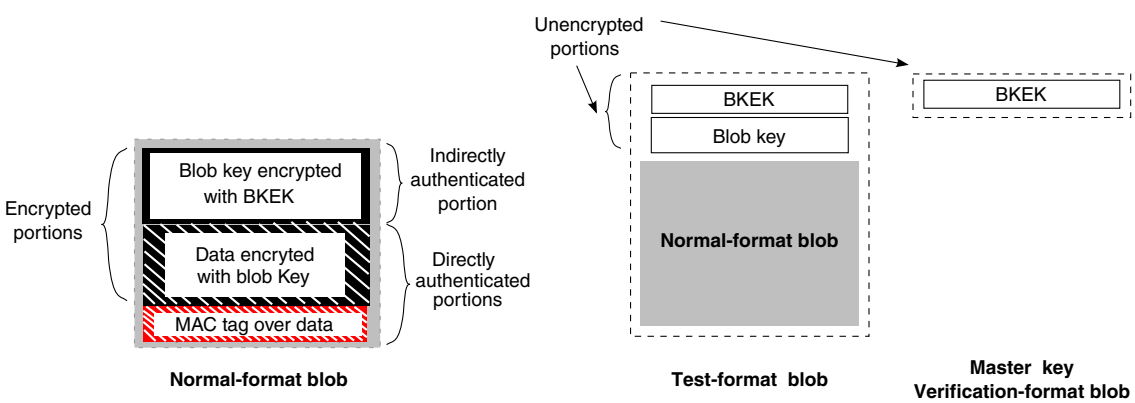


Figure 10-18. Formats of CAAM blobs

- 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 is derived from the secret master key. 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.

10.11.6.5.2 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)).

10.11.6.5.2.1 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.

10.11.6.5.2.2 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 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".)

10.11.6.5.2.3 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.

10.11.6.5.3 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 state 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.

10.11.6.5.4 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.

10.11.6.5.4.1 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.

10.11.6.5.4.2 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 PSDID, SMAPJR and SMAG2/1JR registers of the partition that the blob is being exported from, or imported to. The reason that the PSDID, SMAPJR and SMAGR contents are included in the BKEK derivation for Secure Memory blobs is to cryptographically bind the access permissions to the blob. This ensures that a Secure Memory blob can be imported only into a partition with the same access permission settings (and Security Domain ownership) as the partition from which the blob was exported.

10.11.6.6 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_0000_0000_*xxxx*h, where *xxxx* is the number of bytes of plaintext (maximum length is 65535 bytes), while the initial counter value Ctr0= 0300_0000_0000_0000_0000_0000_0000_0000h. These values are automatically generated during the encapsulation operation.

Figure 10-19 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 10-222). 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.

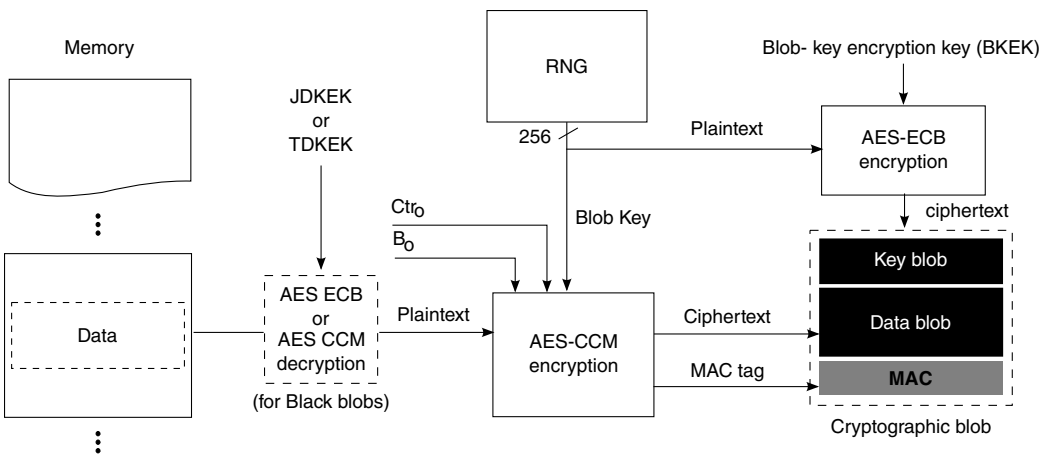


Figure 10-19. Encapsulating a blob

10.11.6.7 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 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 Figure 10-20). 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.

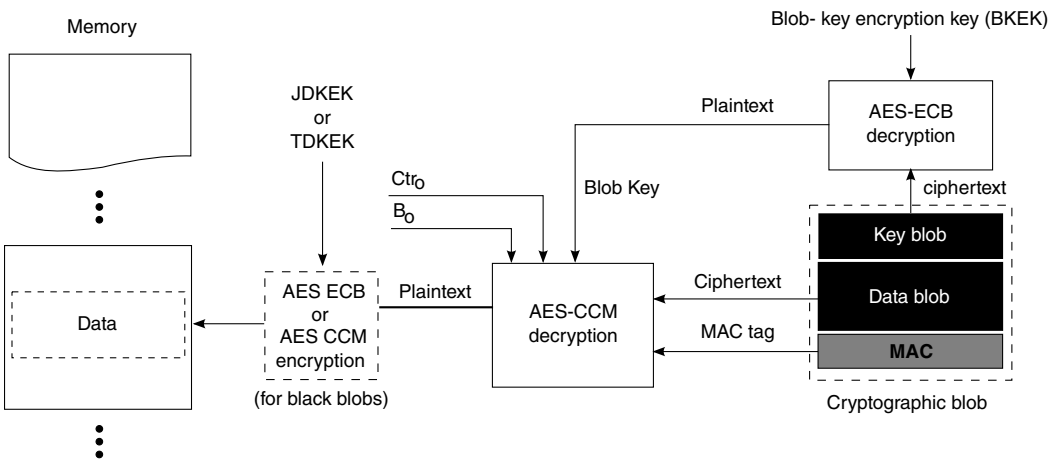


Figure 10-20. Decapsulating a blob

10.11.7 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 entering the FAIL mode. This FAIL mode indicator is an input to CAAM and can be observed via the CAAM Status Register.

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 10-230. Critical security parameters

CSP	Notes	Related cross-reference
Zeroizable master key	Inside of security power island; loaded and locked once at provisioning time	—
CCB Class 1 Key Register	—	See register appendix

Table continues on the next page...

Table 10-230. Critical security parameters (continued)

CSP	Notes	Related cross-reference
CCB Class 2 Key Register	—	See register appendix
PKHA E register	Exponent	See register appendix
Trusted descriptor signing key	Loaded at boot time from RNG	Trusted descriptors
Trusted descriptor key encryption key		
Job descriptor key encryption key	Loaded at boot time from RNG	Keys available in different security modes
ARC4 SBox	—	ARC-4 hardware accelerator (AFHA) CHA functionality
Crypto-engine internal datapath registers	—	See register appendix
Secure Memory CSP partitions	—	Secure memory
Output data FIFO	—	—

10.11.8 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.

10.11.8.1 CAAM Secure Memory features

This table lists the features included in CAAM Secure Memory.

Table 10-231. 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

10.11.8.2 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.

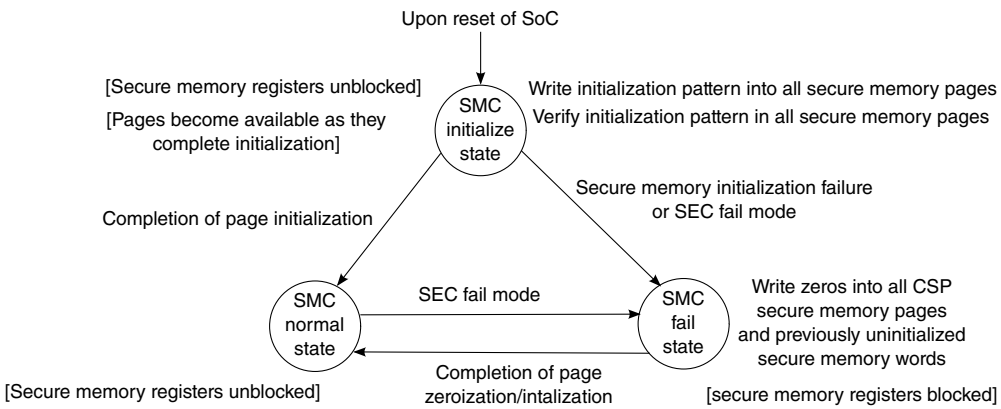


Figure 10-21. Secure memory state machine diagram

10.11.8.2.1 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 asserts the CAAM security violation signal and transitions to SMC fail state. Secure memory also transitions to SMC fail state if CAAM enters CAAM fail mode.

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.

10.11.8.2.2 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 APregister.)

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 DID. Processors operating with the Job Ring 0 owner's DID 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.

10.11.8.2.3 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.

10.11.8.3 Secure memory organization

As shown in Figure 10-22, 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 software entity (e.g. process) is considered a Job Ring owner if the entity can read and write the registers in the Job Ring's register page. Since the ability to read and write a particular CAAM register page is restricted by the CPU's MMU, the operating system (and perhaps also a hypervisor) can determine the Job Ring owners via appropriate MMU settings. Note that CAAM imposes additional access restrictions on access to register pages by examining the DID value that is asserted during register accesses. For each Job Ring there is a JRaDID register in register page 0 that is used to specify a DID value that must be asserted when accessing the registers within a particular Job Ring page. The Job Ring owner can change the access control permissions associated with the Secure Memory partition, and can adjust the amount of memory associated with the partition by adding or releasing pages. Secure Memory is 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.

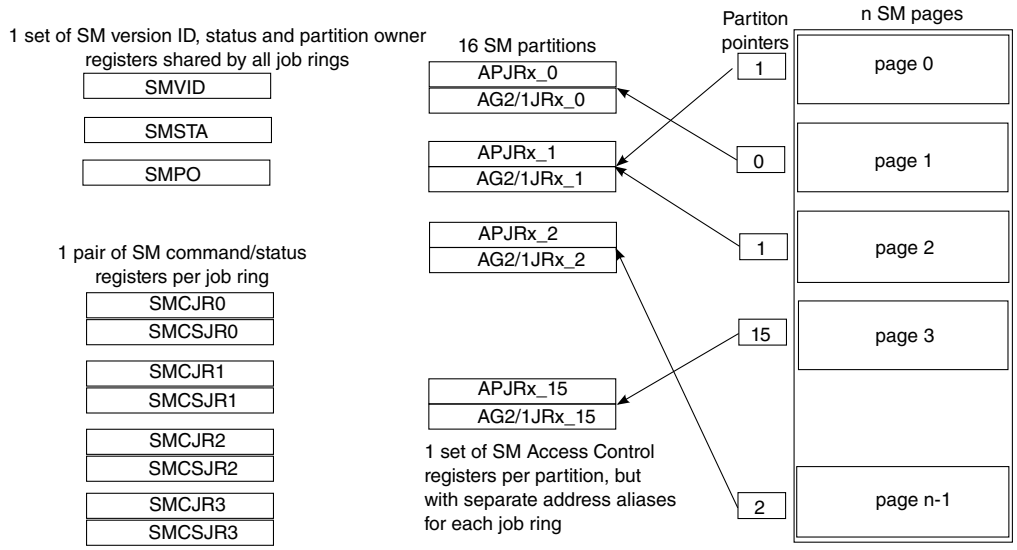


Figure 10-22. Secure memory registers and memory organization

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.

10.11.8.4 Secure memory security functions

The Secure Memory's primary security functions are as follows:

- Automatic RAM zeroization
- Secure Memory Access Control
- Cryptographic protection of exported data

10.11.8.4.1 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.

10.11.8.4.1.1 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 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.

10.11.8.4.2 Secure Memory 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.

10.11.8.4.2.1 Access control through the OS or hypervisor

CAAM assumes that the OS or hypervisor sets the CPU MMU so that only the correct processes have access to specific pages of Secure Memory. If DMA engines can access Secure Memory, the OS or hypervisor must either set a system MMU so that DMA engines can access specific pages of Secure Memory only when they are acting on behalf of the correct processes, or the OS or hypervisor must verify the addresses given to the DMA engines before initiating DMA operations.

10.11.8.4.2.2 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.

10.11.8.4.2.3 Setting Secure Memory 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 TrustZone and domain identifiers (DIDs) can access the memory assigned to this partition:

- If a bus master (including CAAM) accesses a partition using a TZ/DID combination 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 (SMAP) Register.
- If the access is permitted by the SMAG registers, the partition's SMAP 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 access permissions for the partition are established by writing into an unowned partition's SMAP register by means of one of the Job Ring register pages or CAAM register page 0. At that time the SDID value in the Job Ring's JRaDID register is copied to the partition's PSDID register to indicate that the partition is owned by the SDID. Thereafter, the SMAP and SMAG registers of that partition are writable through a Job Ring page only if that Job Ring's JRaDID register has the same SDID value. Note that if the Job Rings have been reassigned by the hypervisor this could be a different Job Ring than the one that claimed the partition. Note also that a partition can be deallocated or pages allocated to or deallocated from a partition by writing to the SMC register in a Job Ring page, but only if the JRaDID register of that Job Ring has the same SDID value as that of the partition. The SMAP, SMAG and SMC registers in CAAM register page 0 can also be used (typically by the hypervisor or TrustZone SecureWorld) to claim or release any partition, or allocate pages to or deallocate pages from any partition, regardless of the partition's SDID value (except that partitions owned by TrustZone SecureWorld can be controlled only by SecureWorld). Note that TrustZone SecureWorld can claim a partition for itself by writing to the partition's SMAP register

via a Job Ring page owned by SecureWorld or via the Secure Memory registers in CAAM register page 0. A Job Ring is claimed for SecureWorld by writing a 1 to the TZ_OWN bit in the Job Ring's JRaDID register. The TZ_OWN bit can be set to 1 only when JRaDID is written via a SecureWorld bus transaction. If the Job Ring is owned by SecureWorld, the PRIM_TZ bit can be set to 1 to indicate that only SecureWorld can access registers in that Job Ring's register page. (Note that PRIM_TZ can be set to 0 even if TZ_OWN is 1, which would permit non-SecureWorld to access that Job Ring's register page. But if PRIM_TZ is 1, nonSecureWorld cannot alter that Job Ring's ownership.) When a partition is claimed via a Job Ring page the TZ_OWN value is copied into the partition's PSDID register along with the SDID value (possibly modified, as described in JRaDID_MS[SDID_MS]). If TZ_OWN is 1 when the partition is claimed, the partition is owned by SecureWorld.

The transaction types that may be specified in the SMAP Register are as follows:

- Read
- Write
- Trusted descriptor override
- Secure memory blob export/import permitted (even if read or write is prohibited)

The SMAP Register also includes some other information about the partition, such as a key modifier field (see Blob encapsulation), lock bits for the SMAP, 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 enabled security events.

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 SMAP 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 SMAP Register also contains a SMAP_LCK bit. Once this has been set to 1, the only field in the SMAP register that can be written is 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.

10.11.8.4.3 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.

10.11.8.4.3.1 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 SDID 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.

10.11.8.4.3.2 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.

10.11.8.5 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 SDID 0.

The SMAPJR and SMAG2/1JR registers for partition 0 are set to allow any type of access by any SDID. 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.

If the desired access control policy requires more than one set of access privileges then additional Secure Memory partitions must be initialized. Because all pages are allocated to partition 0 at POR some of partition 0's Secure Memory pages must be reallocated to other partitions. To do this partition 0's owner or the hypervisor or TrustZone SecureWorld must either de-allocate one or more pages from partition 0 one page at a time, or release the entire partition, which frees all of its pages at once. De-allocating pages or partitions, or allocating pages to partitions, is accomplished by writing to the Secure Memory Command Register. Only the owner of the partition or the hypervisor or TrustZone SecureWorld can deallocate pages from that partition, or release that partition. Any Job Ring owner or the hypervisor 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 security domain identifiers.

10.11.9 Manufacturing-protection chip-authentication process

The manufacturing-protection authentication process is used to authenticate the SoC to the OEM's server. This authentication process ensures that the SoC:

- Is a genuine NXP part
- Is the correct part type
- Has been properly configured by means of fuses
- Is running authenticated OEM software
- Is currently in the secure or trusted mode

Software running on the SoC attests to all this by signing a Manufacturing Protection message using the ECDSA private key stored in the MPPKR. Software running on the SOC then sends this MP attestation message to the OEM's server. The OEM's server verifies the signature over the MP message using the corresponding ECDSA public key.

During the SoC's secure boot process the boot ROM code uses the Super Root Key Hash (SRKH) stored in the SoC's fuse bank to verify a public key that is then used to verify the signature over the device's boot image. The boot ROM code is trusted, and cannot be

altered in a fielded chip, thus successful completion of secure boot verifies that the booted image was signed by a private key whose matching public key was included in the SRKH value. But this alone does not verify that the OEM's public key was included when generating the SRKH. The SRKH may have been burned into fuses by a contract manufacturer, who could have created a SRKH using a public key matching the contract manufacturer's own private key. Thus the contract manufacturer could fool the boot ROM code into booting software written by the contract manufacturer. The Manufacturing Protection authentication procedure is designed to thwart this.

The OEM initiates the Manufacturing Protection authentication procedure by obtaining, via a trusted delivery procedure, one or more genuine SoCs of the correct part type. The OEM creates a public key signature keypair, uses the private key to sign the OEM's software and hashes the public key to yield the Super Root Key Hash. The OEM then burns the SRKH into the fusebank on one or more of the genuine parts. The OEM next boots the part with the OEM's software and runs the MPPrivKey command. One of the inputs to the MPPrivKey command is the SRKH that is in the SoC's fusebank. The MPPrivKey command creates an ECDSA private key derived from the SRKH value and several secret values built into the SoC. The MPPrivKey command leaves the private key in the Manufacturing Protection Private Key register (MPPKR), which cannot be read or altered now that the MPPrivKey command has been executed. The OEM then runs the MPPubKey command, which generates an ECDSA public key that matches the ECDSA private key in the MPPKR. The MPPubKey command exports the public key, which the OEM then stores on the OEM server for use in verifying Manufacturing Protection attestation messages. Generating the MP public key need only be done once. Although no one, not even the OEM, ever sees the Manufacturing Protection private key, the OEM can be assured that it matches the private key in the MPPKR, and that it is cryptographically bound to the SRKH.

When SoCs are delivered to the contract manufacturering facility, the contract manufacturer burns a SRKH hash into fuses, and boots the SoC. When the SoC boots up, the boot ROM code runs the MPPrivKey command, which leaves the ECDSA private key in the Manufacturing Protection Private Key register. Trusted software should also collect SoC-specific information, such as the chip UID and fuse configuration and store them in the Manufacturing Protection Message register (MPMR), and then write-lock the MPMR. (Note that MPMR must be written and locked *before* RNG instantiation -- RNG instantiation prevents writes to the MPMR.) If the contract manufacturer has burned the correct OEM-supplied SRKH into fuses, the OEM-supplied boot image can be executed. This software will create a Manufacturing Protection attestation message, possibly containing additional SoC-specific information. The OEM software will execute the Manufacturing Protection Sign command, which will prepend the data in the MPMR to the attestation message, and then sign the entire message using the ECDSA private key stored in the MPPKR. Note that the MP Sign command can be executed by untrusted

software. This software cannot alter or omit the data that was previously stored in the MPMR by the trusted boot software, so the data in this portion of the attestation message can be trusted to be accurate. The software will then send the MP attestation message to the OEM's server, where it will be verified using the OEM's MP Public Key. If the signature over the attestation message verifies, the OEM can be assured that the message was signed within an SOC that has the correct SRKH burned into fuses, and therefore could only have booted software signed by the OEM's software signing key. The OEM can therefore trust the SoC UID value in the message (because this was stored in the MPMR by trusted boot software), and can be assured that the message originated on a specific SoC. Relying on this assurance, the OEM server can safely download over an SSL connection the confidential information (e.g. proprietary software) needed for operation of the manufactured device. If the contract manufacturer did not burn the proper SRKH into the fusebank bogus software could be booted on the SoC, but the proper MP private key will not have been generated on the SoC, and the MP attestation message will fail the signature verification at the OEM's server. If the contract manufacturer's bogus software never contacts the OEM's server, the SoC will not download the confidential information needed for operation of the manufactured device. This information is input to a key derivation function, to create a private ECC key that is available only to the crypto hardware. The public ECC key can be generated and used to later authenticate the chip and verify the security status of the chip. These properties can be verified by digitally signing a message using this private ECC key. The message may be verified by a server using the public ECC key. Because only a genuine NXP part, configured correctly, and running in the proper security state can correctly sign the message, assurance of all of this is provided by the verification of the message signature. The message cannot be spoofed by untrustworthy software, because the private-key generation, public-key generation and signature functions are all implemented in hardware, and the chip-specific data is supplied by secure-boot firmware. After the signature over the message has been verified, the server can be assured that it is safe to download proprietary data to the chip over a secured connection. The Manufacturing Protection public key signature authentication process takes place in three stages, implemented via the MPPrivK, MPPubK and MPSign functions built into the CAAM hardware.

Table 10-232. Manufacturing-protection chip-authentication functions

Function name	Abbreviation	Authentication steps implemented by function	Cross-reference
Manufacturing-protection private-key generation function	MPPrivK	• Takes input data to be authenticated and hashes that data with a secret value embedded in the silicon. The result is an ECDSA private	MPPrivk_generation function

Table continues on the next page...

Table 10-232. Manufacturing-protection chip-authentication functions (continued)

Function name	Abbreviation	Authentication steps implemented by function	Cross-reference
		key that is securely stored in the MPPKR.	
Manufacturing-protection public-key generation function	MPPubK	• Generates an ECDSA public key that matches the private key in MPPKR and outputs that public key.[1]	MPPubk_generation function
Manufacturing-protection sign function	MPSign	• Takes the value in the MPMsg register and concatenates any additional data supplied as ordinary input to the MPSign function • Signs the concatenated message data using the private key that was stored in the MPPKR by the MPPrivK Generation function • Outputs the signed message, along with the message representative. Software running on the chip sends this signed message to the OEM's server, which then verifies the signature by means of the public key output earlier by the MPPubK Generation function.	MPSign function

1. The MPPubk_generation function is run once on a single chip at the OEM's facility, and the OEM's server retains a copy of this public key to be used later in the authentication process.

10.11.9.1 Providing data to the manufacturing-protection authentication process

The purpose of the manufacturing-protection authentication process is to authenticate certain information, such as the chip's part number, serial number, and the super root key hash, by signing it with a private key that can be used only in a legitimate NXP chip of the correct type running in the secure or trusted states.

The following sections describe how data is input to the manufacturing-protection process.

10.11.9.1.1 Specifying the ECC domain curve for the manufacturing-protection functions

All of the Manufacturing Protection functions must use the same ECC domain curve. This is specified via the CSEL field in the function's PDB. An error will be generated if different curves are specified in different Manufacturing Protection functions. The encoding of the CSEL field is shown below.

Table 10-233. Encoding of the CSEL field

20-17
CSEL (Curve Select) • 0011 = P256 • 0100 = P384 • 0101 = P521 All other values are reserved.

10.11.9.1.2 Providing data to the MPPrivk_generation function

The MPPrivk_generation function is expected to be run only by the secure boot firmware.

10.11.9.1.3 Providing data to the MPPubk_generation function

The only inputs to the MPPubk_generation function are the manufacturing protection private key from the MPPKR, and the elliptic curve selection from the CSEL field in the PDB. The hardware guarantees the correctness of both of these inputs because the MPPKR is accessible only to hardware, and the value in the CSEL field must match the value used in the MPPrivk_generation function.

10.11.9.1.4 Providing data to the MPSign function

To provide data to the MPSign function, trusted software writes some or all of the data into the MPMR, then locks it by setting the MPMRL bit in the Security Configuration Register. Additional data can be provided as ordinary message input to the MPSign function. This additional data will be appended to the content of the MPMR before the data is hashed and signed. All this data is authenticated as having originated from a legitimate NXP chip of a specific type, because the data is signed with the manufacturing-protection private key when the MPSign function is invoked. Note that the MPSign function is intended to be invoked by software whose signature has been verified against the SRKH, however the SRKH has not yet been authenticated. (Authenticating the SRKH is the reason for executing first the MPPrivk_generation function and later the MPSign function.) Until the SRKH authentication process is complete the extra data supplied as ordinary message input to the MPSign function should be treated skeptically.

NOTE

On this device, RNG Instantiation prevents ability to write MPMR. If using manufacturing protection features, MPMR should be written and locked before RNG is instantiated.

10.11.9.1.5 Role of the ROM-resident secure boot firmware

Because the ROM-resident secure boot firmware is the only software that is known to be trusted prior to authentication, it plays a crucial role in the manufacturing-protection authentication process. The ROM-resident boot firmware reads fuse-resident data that needs to be authenticated and either supplies some or all of it as data to the MPPrivK generation function, with trusted software (but not boot ROM code) writing any additionally required data to MPMR. Note that all of the data needed to authenticate the software image that will be booted must be supplied by the ROM-resident firmware using either the MPPrivK generation function or by trusted software configuring the MPMR.

The MPSign function is intended to be invoked by untrusted software that has just booted, which is why the data to be authenticated via the MPSign function must be supplied in advance by trusted ROM-resident secure boot or other trusted software and then securely conveyed to the MPSign function via the MPMR. After the operating system has booted and is able to run a network-protocol stack, application software establishes a communication session with the OEM's server. The application software then runs a descriptor that invokes the MPSign protocol, which uses the ECDSA private key stored in MPPKR to sign a message composed of the content of MPMR followed by other optional data. Note that this additional optional data is supplied by potential untrustworthy software, so it can be relied on only if data authenticated via the MPPrivk_generation function or via the MPMR has demonstrated that the software that supplied the data was properly authenticated via the super root key hash.

10.11.9.2 MPPrivk_generation function

The MPPrivk_generation function uses supplied input data together with a secret value embedded in the silicon to generate an elliptic-curve DSA private key. The function stores the private key in the MPPKR and then the MPPKR is locked to prevent reading or writing from the register bus. The private key is later used in the MPPubk_generation function and the MPSign function. Note that an error is generated if the MPPrivK Generation function is run a second time in the same power-on session.

10.11.9.2.1 Differences between the MPPrivk_generation function and the DL KEY PAIR GEN function

The MPPrivK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPrivK generation function generates only ECDSA private keys, not DSA keypairs.
- The MPPrivK generation function generates the private key by applying a key generation function to the input message data and a secret value embedded in the silicon. The secret value is different in each chip type. The DL KEY PAIR GEN function cannot use the secret value embedded in the silicon.
- The MPPrivK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the CSEL field in the PDB. The DL KEY PAIR GEN function uses curve parameters supplied via the PDB.
- The MPPrivK generation function keeps the private key secret by storing it in the MPPrivK register. The DL KEY PAIR GEN function outputs the private key to memory (along with the public key).

10.11.9.2.2 MPPrivk_generation function parameters and operation

This table describes the MPPrivk_generation parameters.

Table 10-234. MPPrivk_generation function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	2*L	ECC curve parameters.
$G_{x,y}$	Built-in	2*L	Generator point
m	Input	-	The message data to be input to the private-key generator function
s	Stored in MPPKR	N	Private key

This table describes the inputs, outputs and operation of the MPPrivK function.

Table 10-235. MPPrivk_generation function inputs, outputs, and operation

Property	Value
Inputs	• Message data to be input to the private key generation function. • The Csel field in the PDB, selecting a predefined ECC curve.

Table continues on the next page...

Table 10-235. MPPrivk_generation function inputs, outputs, and operation (continued)

Property	Value
Outputs	• The manufacturing protection private key *s*, which is stored in the MPPrivK register.
Operation	• Generate a private key *s*, in the range 1≤s<r. (Hash the supplied message data and the built-in secret value to yield *s*. If *s*=0, alter the input to the generation function by a constant and generate a new *s*.) • Store *s* in MPKeyR as the private key.

10.11.9.2.3 Protocol data block (PDB) for the MPPrivk_generation function

This figure shows the PDB for the MPPrivk_generation function.

Table 10-236. MPPrivk_generation PDB

SGF[1] (bits 31..28)	Reserved (bits 27..21)	Csel (bits 20..17)	Reserved (bits 16..0)
Pointer to m (SGF in bit 31)			
Message length constant (Not a pointer. No SGF)			

1. SGF (Scatter Gather Flag) If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.

The encoding of the CSEL field is shown in Table 10-233.

10.11.9.3 MPPubk_generation function

The MPPubk_generation function uses the private key value stored in the MPPrivK register by the MPPrivk_generation function to generate a matching elliptic-curve DSA public key. The curve selected via the Csel field in the PDB must match the curve used by the MPPrivk_generation function, else an error is generated. The public key created by the MPPubk_generation function is written out to the specified results destination. Note that the MPPubK Generation function is intended to be run just once, at the OEM's facility, but no harm is done if it is run at other times.

10.11.9.3.1 Differences between the MPPubk_generation function and the DL KEY PAIR GEN function

The MPPubK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPubK generation function generates only an ECDSA public key, not DSA or ECDSA keypairs.

- The MPPubK generation function creates a public key to match the private key value that was stored in the MPPrivK register by the MPPrivK generation function.
- The MPPubK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the Csel field in the PDB.
- The MPPubK generation function outputs only the public key. Unlike the DL KEY PAIR GEN function it does not output the private key.
- The private key stored in the MPPrivK register by the MPPrivK generation function is not altered, and remains available for use in the MPSign function.

10.11.9.3.2 MPPubk_generation function parameters and operation

This table describes the MPPubk_generation parameters.

Table 10-237. MPPubk_generation function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	2*L	ECC curve parameters.
$G_{x,y}$	Built-in	2*L	Generator point
s	Read from MPPKR	N	Private key
$W_{x,y}$	Output	2*L	Public key

This table describes the inputs, outputs and operation of the MPPubK function.

Table 10-238. MPPubk_generation function inputs, outputs, and operation

Property	Value
Inputs	• The Csel field in the PDB, selecting a predefined ECC curve. • The manufacturing protection private key *s*, which is read from the MPPrivK register
Outputs	• The manufacturing protection public key $W_{x,y}$, which is output to memory.
Operation	• Compute $W_{x,y} = sG_{x,y}$ • Output $W_{x,y}$ as the public key.

10.11.9.3.3 Protocol data block (PDB) for the MPPubk_generation function

This figure shows the PDB for the MPPubk_generation function.

Table 10-239. MPPubk_generation PDB

SGF[1] (bits 31..28)	Reserved (bits 27..21)	Csel (bits 20..17)	Reserved (bits 16..0)
Pointer to Wx,y (SGF in bit 31)			

1. SGF (Scatter Gather Flag) If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.

The encoding of the CSEL field is shown in Table 10-233.

10.11.9.3.4 Running the MPPubK generation function at the OEM's facility

When a chip is first adopted by an OEM, the OEM runs the MPPubk_generation function on a sample of the chip and saves the public key of the manufacturing protection keypair on the OEM's server. Running the MPPubk_generation function at the OEM's facility this one time guarantees that the public key is authentic, that is, that it matches the private key that is used to sign messages generated by properly configured NXP chips of this type. The OEM will first have programmed the trusted root public key into fuses, and then reboot the chip. The secure boot firmware will run the MPPrivk_generation function at POR and store the manufacturing protection private key in the MPPrivK register. The MPPubk_generation function will read the manufacturing protection private key from the MPPrivK register and generate a matching public key. Later when the identically configured chips are booted within the contract manufacturing facility, the identical private key will be generated by the MPPrivK generation function. The MPPrivk_generation function stores the private key in the MPPKR for use in the MPSign function. The message signed by the MPSign function can be authenticated against the manufacturing protection public key stored on the OEM's server.

10.11.9.4 MPSign function

MPSign is the elliptic-curve, digital-signature algorithm (ECDSA) signing function used in the manufacturing protection authentication process. See Manufacturing-protection chip-authentication process for a discussion of this process. MPSign supports only ECDSA in prime fields. This function takes message data as input, and outputs a signature over a message composed of the content of the MPMR, followed by the input-data message.

Note that the curve specified via the Csel field in the PDB must match the curve used in the MPPrivk_generation function. This table lists the MPSign protocol parameters.

10.11.9.4.1 MPSign function parameters and operation

This table describes the MPSign function parameters.

Table 10-240. MPSign function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	2*L	ECC curve parameters
$G_{x,y}$	Built-in	2*L	Generator point
s	Read from MPPKR	N	Private key
m	Input	-	The message data to be signed.
C	Output	N	First part of digital signature
d	Output	N	Second part of digital signature. The buffer for *d* must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding.
mes-rep	Output	256	The hash of the MPMR concatenated with m.

This table describes the inputs, outputs and operation of the MPSign function.

Table 10-241. MPSign function inputs, outputs, and operation

Property	Value
Inputs	• m, the message data to be signed • u, the private key (from the MPPrivK register) • a,b, the curve parameters (selected via the Csel field in the PDB)
Outputs	• The signature over the signed message. • mes-rep, the hash of MPMR concatentated with the message data
Operation	• Compute $V_{x,y} = u\ G_{x,y}$, $c = V_x \bmod r$. If c=0, try again with a new *u*. • Compute $d = u^{-1}(f+sc) \bmod r$. If d=0, try again with a new *u*. • Output (*C*, *d*) as the signature.

10.11.9.4.2 Protocol data block (PDB) MPSign function

This figure shows the MPSign function PDB.

Table 10-242. MPSign function PDB

SGF[1] (bits 31..28)	Reserved (bits 27..21)	Csel (bits 20..17)	Reserved (bits 16..0)
Pointer to m (SGF in bit 31)			
Pointer to mes-rep (SGF in bit 30)			
Pointer to c (SGF in bit 29)			
Pointer to d (SGF in bit 28)			
Message length constant (Not a pointer. No SGF)			

1. SGF (Scatter Gather Flag) If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.

The encoding of the CSEL field is shown in Table 10-233.

10.12 CAAM service error detection, recovery (reset), and reconfiguration

CAAM can be concurrently used by multiple users through use-case optimized Service interfaces that efficiently coordinate the use of CAAM hardware resources and address different user needs. The sharing of CAAM hardware is generally managed through privileged software providing means to assign physical or virtualized CAAM service access to less privileged software. While this use model enables efficient sharing of hardware resources, it requires that errors introduced by one user must not significantly affect other users.

CAAM addresses this requirement by enabling service interface users to handle most of their own service management needs and by enabling management software to monitor and, if necessary, take corrective action if individual CAAM users do not 'behave' in a cooperative manner, have fatally failed and need to be terminated or restarted, or simply have finished and their assigned CAAM service access capability can be made available to other users. In addition, management software may detect fatal error conditions within SoC hardware components and requires support for a limited reset of CAAM and/or the connected components to maximize SoC operation availability, e.g., perform a sub-system reset to minimize SoC out-of-service times.

The following sections describe CAAM's facilities available to both (ordinary) user and (privileged) management software to detect CAAM service problems and take corrective action, or enable and reassign CAAM services to alternate users.

10.12.1 Software CAAM Reset

The software CAAM reset will cause most registers and state machines in SEC to reset. Registers that are not affected by software reset are listed in the description of the SWRST field in the Master Config Register (MCFGR). A software CAAM reset is initiated by writing 1 to the SWRST bit. 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 LARGE_BURST, AXIPIPE, AWCACHE and ARCACHE, so to avoid disrupting outstanding DMA transactions when initiating a SWRST, these fields should be rewritten with their current values.

10.12.2 Job ring error detection, recovery, reset and reconfiguration

CAAM's Job Ring interface can be independently assigned (and re-assigned) to different users. Examples include, but are not limited to, Arm TZ software, operating systems, and (ordinary) user processes. In all cases privileged software is either using a Job Ring itself or is granting Job Ring use to less privileged software.

The following sections describe the error detection, recovery, reset features available to less privileged (e.g. user mode) software being granted use of the Job Ring (the 'user') and the features available to the privileged software having granted Job Ring use to the user (the 'manager'). A 'manager' may need to utilize Job Ring services and thus also play the role of a 'user'.

10.12.2.1 Job ring user error detection, recovery, reset, and reconfiguration services

This section describes the error detection, recovery, reset, and reconfiguration features available to Job Ring users. Error detection, recovery, reset, and reconfiguration features available to privileged system management software are described in section Job ring error detection, recovery, reset, and reconfiguration management services.

10.12.2.1.1 Error recovery

The functional errors that can be detected by Job Rings are the ones that are reported in the ERR_TYPE field of the Job Ring Interrupt Register (JRINTR). Almost all of these errors are caused by incorrect programming of the Job Ring.

- 00001b - Memory access error writing status to Output Ring
- 00011b - Invalid Input Ring Base Address Register IRBAR (not on a 4-byte boundary)
- 00100b - Invalid Output Ring Base Address Register ORBAR (not on a 4-byte boundary)
- 00101b - Invalid write to IRBAR or Input Ring Size Register (IRSR)
- 00110b - Invalid write to ORBAR or Output Ring Size Register (ORSR)
- 00111b - Job ring reset released before Job Ring is halted.
- 01000b - Removed too many jobs (ORJRR larger than ORSFR)
- 01001b - Added too many jobs (IRJAR larger than IRSAR)
- 01010b - Writing ORSFR > ORSR
- 01011b - Writing IRSAR > IRSR

- 01100b - Writing ORWIR > ORSR in bytes
- 01101b - Writing IRRIR > IRSR in bytes
- 01110b - Writing IRSAR when ring is not empty
- 01111b - Writing IRRI when ring is not empty
- 10000b - Writing ORSFR when ring is not empty
- 10001b - Writing ORWIR when ring is not empty

Each of these errors cause the JRI bit, if IMSK is not set in the Job Ring Configuration Register (JRCFGR), the JRE bit, and the ERR_TYPE field to be set in the Job Ring Interrupt Register (JRINTR). Also, if IMSK is not set, the CAAM interrupt output for this Job Ring will assert.

The recovery procedure for error type 00011b is to reprogram the Input Ring Base Address Register with an address that is aligned to a 4-byte boundary. Then clear the JRI and JRE bits in the Job Ring Interrupt Register.

The recovery procedure for error type 00100b is to reprogram the Output Ring Base Address Register (ORBAR) with an address that is aligned to a 4-byte boundary. Then clear the JRI and JRE bits in the Job Ring Interrupt Register.

The error codes 01010b, 01011b, 01100b, and 01101b are caused by attempting to write out of range values to the Output Ring Slots Full Register (ORSFR), Input Ring Slots Available Register (IRSAR), Output Ring Write Index Register (ORWIR), or Input Ring Read Index Register (IRRIR). The write operation to the ORSFR, IRSAR, ORWIR, or IRRIR register that caused the error will be ignored. In order to recover from these errors, simply clear the JRE and JRI bits in the Job Ring Interrupt Register and write a valid value in the ORSFR, IRSAR, ORWIR, or IRRIR register. If a valid value cannot be determined, a Job Ring reset may be required.

All other error types are more serious and may cause some jobs to never run or cause loss of some completed job information. The recovery procedure for these errors is perform a Job Ring reset (see Job Ring Command Register JRCR RESET field). A Job Ring reset will clear all registers for that particular Job Ring except the REIRxJRa, IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers. The REIRxJRa registers should be manually reset after a Job Ring reset. The IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers can be rewritten or not, as appropriate, after a Job Ring reset. If a Job Ring reset cannot be performed for some reason, then a management service will be required.

10.12.2.1.2 Unrecoverable conditions

An unrecoverable condition is one from which the Job Ring user will not be able to recover or will not be able to recover without assistance from the manager or from the user of one of the other Job Rings. A security violation is an unrecoverable error condition.A second type of unrecoverable error condition occurs when a privileged CAAM manager has stopped CAAM to prevent further jobs from processing.

If CAAM detects a security violation, this will be reported in the Job Ring Interrupt Register (JRINTR). The ENTER_FAIL bit in the JRINTR sets when security violation occurs and the EXIT_FAIL bit in that same register sets when the security violation is cleared. If the interrupt is not masked and the FAIL_MODE bit is set in the JRCFGR, the interrupt for this Job Ring will assert when the ENTER_FAIL bit sets and again when the EXIT_FAIL bit sets. Any jobs running while the security violation is active terminate with a DECO fail mode error status (see Job termination status/error codes). If the FAIL_MODE bit in JRCFGR is not set when the security violation is detected, the Job Ring will begin to halt. During the time while jobs from this Job Ring are still in progress, the HALT field in the JRINTR will return 01b, indicating that the Job Ring is halting. When all jobs are complete, the HALT field of the JRINTR will return 10b.

Until the security violation is resolved, the Job Ring will remain halted or will continue to process jobs, but return them with a fail mode error status, depending on the setting of the FAIL_MODE bit in JRCFGR. After the security violation is resolved, the job ring will process jobs normally again. However, certain features of CAAM may remain disabled until the manager performs a software CAAM reset or a power on reset.

If the manager stops CAAM either as a result of a recoverable error interrupt or to begin reading various debug registers, no further jobs for any user will run until the manager restarts CAAM to allow processing to resume.

10.12.2.1.3 User reconfiguration options

It is possible for the user to reconfigure the Job Ring. This can be done by performing the following steps:

- Stop adding new jobs to the input ring.
- Wait until all jobs issued to the input ring are complete or perform a Job Ring reset or park and wait for halt. A park (initiated by writing the PARK bit of the JRCR register) will allow the jobs to complete normally, while a reset will cause the jobs to terminate with a DECO Job Ring reset error.
- Process all jobs in the output ring.
- Clear any interrupt or error status in the Job Ring Interrupt Register (JRINTR).
- Clear the recoverable error indication registers (REIRxJRa).
- Reprogram the Job Ring (IRBAR, IRSR, ORBAR, ORSR, and JRCFGR) registers and issue new jobs to the input ring.

10.12.2.2 Job ring error detection, recovery, reset, and reconfiguration management services

This section describes the error detection, recovery, reset, and reconfiguration features available to privileged system software to manage Job Ring users and their assigned resources. Error detection, recovery, reset, and reconfiguration features available to Job Ring user software are described in section Job ring user error detection, recovery, reset, and reconfiguration services.

10.12.2.2.1 Recoverable error status notifications

The Job Ring recoverable error indication registers (REIR0JRa, REIR2JRa, REIR4JRa, and REIR5JRa) provide additional information in the event of a memory access error reading or writing the Job Ring, or while one of the Job Ring jobs is executing.

In the event of a memory access error while writing the output ring, error status will also be reported in the Job Ring Interrupt Status register (JRINTR). In the event of memory access errors reading the input ring, reading a Job Ring job descriptor, or during the execution of a Job Ring job descriptor, the error code written to the output ring will indicate a DMA error or an error reading the Descriptor or Descriptor address. See Job termination status/error codes.

10.12.2.2.2 Ring user access termination procedure

When the manager wants to terminate the access for the current ring user, the manager must first prevent the current user from further accessing the Job Ring. Then the manager must reset the Job Ring to terminate any in progress jobs using the Job Ring Command register (JRCR). This also prevents any new jobs that were already programmed in the input ring from starting. When the HALT field in the Job Ring Interrupt Status register (JRINTR[HALT]) indicates that the Job Ring has stopped, perform a Job Ring soft reset using JRCR or, if virtualization is enabled, stop the Job Ring. This procedure does not save any state information for the Job Ring, so the ring will need to be completely reconfigured for the next user.

10.12.2.2.3 Ring user (re-)assignment procedure

If the Job Ring is currently assigned to a user, terminate that user's access as described above. Stop the Job Ring if it is not already stopped, if virtualization is enabled. Program the Job Ring DID register. Start the job ring, if virtualization is enabled. Enable user Job Ring register access so that the user can program the IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers.Set the IRBAR, IRSR, IRSAR, ORBAR, ORSR, ORSFR,

JRCFGR_MS, JRCFGR_LS, IRRIR and ORWIR appropriately. If the Job Ring is being reassigned to a new user, assign the appropriate default values. If the Job Ring is being returned to a previous user, restore the user's saved Job Ring context.

10.12.3 RTIC error detection, recovery, reset, and reconfiguration

CAAM's Run-Time Integrity Checker (RTIC) services are designed to be utilized by privileged software because they may affect all CAAM service users. Privileged RTIC software (e.g., an OS driver) may make select (including no) RTIC services available to unprivileged (ordinary) software. The following sections describe the error detection, recovery, reset features available to privileged software utilizing CAAM's RTIC services.

10.12.3.1 RTIC user services

As stated above, the RTIC feature set is designed to be utilized only by privileged software, i.e., any RTIC services made available to unprivileged software need to be made available by the privileged software. Privileged software may provide a subset (including none) of the available services available to privileged software to the unprivileged software. For a list of the available services see the following RTIC management services section.

10.12.3.2 RTIC management services

The RTIC interface is used by management software to offer both user and privileged run-time integrity checking. Management software can initially use RTIC to perform One-time hash generation, and offer this service to other user software. This is intended for initial boot services only. After boot, the management software is expected to configure RTIC for run-time integrity checking. RTIC will stay in the run-time integrity check mode until a system reset occurs.

There are two run-time services which may be offered: Temporary run-time integrity checking; and permanent run-time integrity checking. Temporary run-time integrity checking can be disabled by software, or will terminate with an error interrupt. This is a recoverable error. Permanent run-time integrity checking cannot be disabled, and will run until a system reset occurs, or an error is detected. An error will be reported as a hardware security violation.

RTIC can monitor, or hash, up to four independent regions of memory. The management software can offer four services simultaneously. The completion of one monitoring, or hashing, operation does not affect the operation of those other three services. When RTIC is not monitoring, or hashing, any of the four memory regions, it can be configured for another operation, independently of any other operations it is currently performing.

10.12.3.2.1 Recoverable error conditions

The only error condition for one-time hash generation is a bus error; an access to memory is not allowed. This may be caused by an invalid address, or perhaps a hardware error (e.g., ECC error). When an interrupt is generated, manager software will read the RTIC status register to determine whether the hash generation completed properly, or terminated with a bus error. In either case, after the one-time hash generation has completed, RTIC can be re-configured to perform a new operation.

Temporary run-time integrity checking can be disabled by management software, or it will receive an interrupt specifying that a memory integrity check, or address error has occurred. In this case, an interrupt can only mean that an error has occurred, and RTIC will stop hashing that region of memory. Software can determine the cause of the error by reading the RTIC status register. The status register has the status of all four memory regions, and the act of reading the status register clears all errors. RTIC can then be re-configured to perform its next operation or remain idle. The management software will need to deal with the corrupted memory, perhaps by terminating the process being monitored.

10.12.3.2.2 Unrecoverable error conditions

Permanent run-time integrity checking can only terminate with a non-recoverable error. In this mode, RTIC is locked to prevent any software turning off the checking. RTIC will typically be checking a critical piece of software, such as the trusted base component of the manager software itself. This software is expected to be so critical that any detected modification of code memory can only be fixed through a system shutdown and re-boot. In this case, RTIC will report the detected error by issuing a security violation.

After issuing a security violation, RTIC will not be usable until the next system reset. There is no recovery from this error condition.

10.12.3.2.3 Reconfiguration procedure

A recoverable error will disable run-time integrity checking of the memory region that had the error. Management software reads the RTIC status register to determine which memory had a recoverable error. The RTIC configuration for that region will go back to an idle state and can simply be configured for a new monitoring operation.

10.12.4 Global and DECO error detection, recovery, reset, and reconfiguration

CAAM's global and DECO services are designed to be utilized by privileged software because they may affect all CAAM service users. Privileged CAAM software (e.g., an OS driver) may make select (including no) global CAAM or DECO services available to unprivileged (ordinary) software. The following sections describe the error detection, recovery, reset features available to privileged software utilizing CAAM's global and DECO services.

10.12.4.1 Global and DECO user services

As stated above, the global CAAM and DECO services feature set is designed to be utilized only by privileged software, i.e., any global CAAM or DECO services made available to unprivileged (ordinary) software need to be made available by privileged software. Privileged software may provide a subset (including none) of the available services available to privileged software to the unprivileged software. For a list of the available services see the following Global and DECO management services section.

10.12.4.2 Global CAAM and DECO management services

This section describes the error detection, recovery, reset, and reconfiguration features available to manage global CAAM and DECO services and resources.

10.12.4.2.1 Error detection

Descriptors that do not run properly in a DECO can encounter two classes of problems. The first class of problems consists of errors detected by DECO, CCB, or a CHA. In such cases, DECO reacts to the error by terminating execution and returning the corresponding error status. Note that such errors include execution errors, e.g. illegal DECO commands, and functional errors, e.g. bad data or ICV failures. Such errors are handled via normal channels and no additional access to the DECO is required.

The second class of problems is due to a hang. The vast majority of hangs are detected by the watchdog timer. If the watchdog timer fires, then the hang turns into a problem of the first class and no further action is required. However, there are some hangs that the timer either cannot detect or, having detected the hang, from which DECO still can't recover. (The most common case that the watchdog can't detect is a loop that doesn't terminate.) In such cases, software must intervene through the recovery procedure.

Although it is possible to detect that a DECO has hung by noticing that the status for a job has not been returned after an unusually long time, there is a better method. The DECO Availabilty Register (DAR) can be used to detect a DECO that is potentially hung. Each DECO automatically clears its corresponding bit in this register whenever it is idle. (Note that DECOs always report themselves as idle between jobs.) If it is suspected that a job has hung in a DECO, then software can write a 1 to each bit in this register which corresponds to a DECO. (It is safe to write all 32 bits of this register to 1 since unimplemented bits are not written and will remain 0 when read.) Software would then wait long enough for most jobs to finish and then read the register. Any bit position that is still a 1 would correspond to a DECO that has not completed its job. Software can use this method proactively. For example, software could set all the bits in the DECO Availability Register to 1, then wait a second, and check to see if any DECO was not idle during that second. If not, then set all the bits to 1 again and repeat. The actual length of time used between checks is a function of the type of descriptors being run in a particular application, the likelihood that such descriptors will hang, and the availability of a processor to check on progress.

If the DECO Availability Register indicates that a DECO has not completed a job in an excessive amount of time, further work can be done to identify the descriptor running in that DECO by reading debug registers. These registers, which are always accessible for reading, include the DECO Debug Job Register (D0DJR), the DECO Debug DECO Register (D0DDR), the DECO Debug Job Pointer (D0DJP), and DECO Debug Share Pointer Register (D0SDP). They can be used to identify what job and/or shared descriptor is running. Furthermore, the first two of these registers can identify whether progress is being made. This information can help determine whether one of the recovery processes needs to be utilized. If progress is being made, or if this particular job is expected to run for a very long time, then no action needs to be taken.

10.12.4.2.2 Recovery procedure

In order to recover the DECO identified as hung in the DECO Availability Register (DAR), write a 1 to the corresponding bit in the DECO Reset Register (DRR). Note that this forces the DECO into an error state from which that DECO should recover. However, that recovery may not be instantaneous as the DECO needs to wait for outstanding DMA transactions to complete. Once the DECO completes the reset process, the corresponding bit in the DECO Availability Register will clear. The bit in the DECO Reset Register is auomatically cleared.

10.13 CAAM register descriptions

CAAM`s register address space is divided into multiple register address blocks. Each register address block is dedicated to a specific purpose, and access to different register blocks may be restricted to different software processes via the MMU or System MMU or CAAM register access control hardware. Each CAAM register block occupies 4K bytes of address space. The lowest address of each block corresponds to the lowest address of a memory page, whose size is determined by the MMUs in the SoC. Successive CAAM register blocks occupy successive memory pages. In SoCs with 4k byte memory pages, each CAAM register block occupies a full memory page. In SoCs with 64k byte memory pages, each CAAM register block occupies the lowest 4k bytes of the 64k byte memory page, with 60K bytes of unimplemented addresses in the upper portion of the memory page. CAAM`s register address blocks are shown in the table below.

Table 10-243. CAAM Register Address Block Identifiers

Block Identifier	Included registers
0	General registers (e.g., configuration, control, debugging, and RNG)
1-3	Job Ring registers (JR0-2) and Secure Memory registers
6	Real-time integrity check registers
8	Descriptor controller (DECO) and CHA control block (CCB)

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 represent how these registers are accessed over the register bus as 32-bit words.

NOTE

CAAM performs certain actions automatically immediately after POR, and CAAM may be used by the boot firmware at boot time. As a consequence, by the time software reads the CAAM registers their reset values may already have been changed from the POR values.

NOTE

The CAAM address space is divided into 16 4 KB pages to match the access granularity of the MMU. 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 DID-based access control mechanism, or via the CPU's MMU. For instance, many configuration and status registers are located within page 0 and these are intended to be accessed only by privileged software.

The settings in many of these page 0 registers affect the behavior of CAAM hardware for all users of CAAM. The CPU MMUs should be configured to prevent page 0 from being accessed by untrusted software. The CAAM register interface uses bus transaction attributes to limit access to register page 0. In this chip both TrustZone SecureWorld and nonSecureWorld can access these registers, but only from the any master. The registers that control each Job Ring are located in a separate address block 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 address blocks. 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.

NOTE

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 address blocks. 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 unwritable by other Job Ring owners.

NOTE

The reset value of some registers differs between different versions of CAAM. To ensure driver compatibility across different versions of CAAM, when updating fields within registers the registers should first be read, the required fields updated, and then the register should be written. This will avoid inadvertently changing the settings of other fields in the same register.

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 privileged software (e.g. boot software, hypervisor, secure operating system).

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 register name will be associated with 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 register name and the list of addresses at which that register is accessible will be indicated as aliases. 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, these registers are accessed over the register bus as 32-bit words. Note that all registers other than the CCB/DECO registers must be accessed only as full 32-bit words. Byte enables are permitted only for the CCB/DECO registers. All addresses not shown are reserved.

NOTE

In order to facilitate software compatibility across both big-endian and little-endian SoCs, CAAM can be set to double-word swap registers that are indicated in the Memory Map as 64 bits. (See MCFGR[DWT].) These registers hold address pointers or integers larger than 32 bits, so setting the DWT bit appropriately will cause 64-bit read or write transactions to place the most-significant and least-significant words in the proper positions in 64-bit registers. A register whose width is shown as 32 bits in the Memory Map should be accessed as a single 32-bit bus transaction, even if there is an adjacent related 32-bit register (e.g. CAAMVID_MS and CAAMVID_LS). CAAM does not double-word swap the addresses of such register pairs, so accessing them via 32-bit bus transactions will facilitate software portability across big-endian and little-endian SoCs.

Data read from and written to DECO registers by software is treated as control data for the purpose of endianness conversion.

CAAM Secure Memory base address: 0x100000

10.13.1 CAAM memory map

CAAM base address: 3090_0000h

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
4	Master Configuration Register (MCFGR)	32	RW	0008_2300
8	Page 0 SDID Register (PAGE0_SDID)	32	RW	0000_0000
C	Security Configuration Register (SCFGR)	32	RW	0000_0000
10	Job Ring 0 DID Register - most significant half (JR0DID_MS)	32	RW	0000_0000
14	Job Ring 0 DID Register - least significant half (JR0DID_LS)	32	RW	0000_0000
18	Job Ring 1 DID Register - most significant half (JR1DID_MS)	32	RW	0000_0000
1C	Job Ring 1 DID Register - least significant half (JR1DID_LS)	32	RW	0000_0000
20	Job Ring 2 DID Register - most significant half (JR2DID_MS)	32	RW	0000_0000
24	Job Ring 2 DID Register - least significant half (JR2DID_LS)	32	RW	0000_0000
58	Debug Control Register (DEBUGCTL)	32	RW	0000_0000
5C	Job Ring Start Register (JRSTARTR)	32	RW	0000_0000
60	RTIC OWN Register (RTIC_OWN)	32	RW	0000_0000
64	RTIC DID Register for Block A (RTICA_DID)	32	RW	0000_0000
6C	RTIC DID Register for Block B (RTICB_DID)	32	RW	0000_0000
74	RTIC DID Register for Block C (RTICC_DID)	32	RW	0000_0000
7C	RTIC DID Register for Block D (RTICD_DID)	32	RW	0000_0000
94	DECO Request Source Register (DECORSR)	32	RW	0000_0000
9C	DECO Request Register (DECORR)	32	RW	0000_0000
120	DECO Availability Register (DAR)	32	RW	0000_0000
124	DECO Reset Register (DRR)	32	WO	0000_0000
184	Job Ring 0 Secure Memory Virtual Base Address Register (JR0SMVBAR)	32	RW	0010_0000
18C	Job Ring 1 Secure Memory Virtual Base Address Register (JR1SMVBAR)	32	RW	0010_0000
194	Job Ring 2 Secure Memory Virtual Base Address Register (JR2SMVBAR)	32	RW	0010_0000
220	Peak Bandwidth Smoothing Limit Register (PBSL)	32	RW	0000_0000
240	DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS)	32	RO	Table 10-243
244	DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS)	32	RO	Table 10-243
248	DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS)	32	RO	Table 10-243
24C	DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS)	32	RO	Table 10-243
250	DMA0 AXI ID Enable Register (DMA0_AID_ENB)	32	RO	Table 10-243
260	DMA0 AXI Read Timing Check Register (DMA0_ARD_TC)	64	RW	0000_0000_0000_0000
26C	DMA0 Read Timing Check Latency Register (DMA0_ARD_LAT)	32	RW	0000_0000
270	DMA0 AXI Write Timing Check Register (DMA0_AWR_TC)	64	RW	0000_0000_0000_0000
27C	DMA0 Write Timing Check Latency Register (DMA0_AWR_LAT)	32	RW	0000_0000
300 - 33F	Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)	8	RW	00
380 - 39F	Manufacturing Protection Message Register (MPMR0 - MPMR31)	8	RW	00

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
3C0 - 3DF	Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)	8	RO	00
3F8	Manufacturing Protection ECC Register (MPECC)	32	RO	0000_0000
400 - 41C	Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7)	32	RW	Table 10-243
420 - 43C	Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)	32	RW	Table 10-243
440 - 45C	Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)	32	RW	Table 10-243
4E0	Secure Key Nonce Register (SKNR)	64	RW	0000_0000_0000_0000
50C	DMA Status Register (DMA_STA)	32	RO	0000_0080
510	DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)	32	RO	Table 10-243
514	DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)	32	RO	Table 10-243
518	DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)	32	RO	Table 10-243
51C	DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)	32	RO	Table 10-243
524	DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN)	32	RO	Table 10-243
530	DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)	32	RW	0000_0000
534	DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)	32	RW	0000_0000
538	DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)	32	RW	0000_0000
53C	DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)	32	RW	0000_0000
540	DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)	32	RW	0000_0000
544	DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)	32	RW	0000_0000
548	DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)	32	RW	0000_0000
54C	DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)	32	RW	0000_0000
600	RNG TRNG Miscellaneous Control Register (RTMCTL)	32	RW	0000_0000
604	RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)	32	RW	0001_0022
608	RNG TRNG Poker Range Register (RTPKRRNG)	32	RW	0000_09A3
60C	RNG TRNG Poker Maximum Limit Register (RTPKRMAX)	32	RW	0000_6920
60C	RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)	32	RO	0000_0000
610	RNG TRNG Seed Control Register (RTSDCTL)	32	RW	0C80_09C4
614	RNG TRNG Sparse Bit Limit Register (RTSBLIM)	32	RW	0000_003F
614	RNG TRNG Total Samples Register (RTTOTSAM)	32	RO	0000_0000
618	RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)	32	RW	0000_0640
61C	RNG TRNG Frequency Count Register (RTFRQCNT)	32	RO	0000_0000
61C	RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)	32	RW	0000_6400
620	RNG TRNG Statistical Check Monobit Count Register (RTSCMC)	32	RO	0000_0000
620	RNG TRNG Statistical Check Monobit Limit Register (RTSCML)	32	RW	010C_0568

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
624	RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)	32	RO	0000_0000
624	RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)	32	RW	00B2_0195
628	RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)	32	RO	0000_0000
628	RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)	32	RW	007A_00DC
62C	RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)	32	RO	0000_0000
62C	RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L)	32	RW	0058_007D
630	RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C)	32	RO	0000_0000
630	RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L)	32	RW	0040_004B
634	RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C)	32	RO	0000_0000
634	RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L)	32	RW	002E_002F
638	RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC)	32	RO	0000_0000
638	RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL)	32	RW	002E_002F
63C	RNG TRNG Status Register (RTSTATUS)	32	RO	0000_0000
640 - 67C	RNG TRNG Entropy Read Register (RTENT0 - RTENT15)	32	RO	0000_0000
680	RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)	32	RO	0000_0000
684	RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)	32	RO	0000_0000
688	RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)	32	RO	0000_0000
68C	RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)	32	RO	0000_0000
690	RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)	32	RO	0000_0000
694	RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)	32	RO	0000_0000
698	RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)	32	RO	0000_0000
69C	RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)	32	RO	0000_0000
6C0	RNG DRNG Status Register (RDSTA)	32	RO	0000_0000
6D0	RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)	32	RO	0000_0000
6D4	RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)	32	RO	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
6E0	RNG DRNG Hash Control Register (RDHCNTL)	32	RW	0000_0000
6E4	RNG DRNG Hash Digest Register (RDHDIG)	32	RO	0000_0000
6E8	RNG DRNG Hash Buffer Register (RDHBUF)	32	WO	0000_0000
A00	Partition 0 SDID register (P0SDID_PG0)	32	RO	0000_0000
A04	Secure Memory Access Permissions register (P0SMAPR_PG0)	32	RW	0000_00FF
A08	Secure Memory Access Group Registers (P0SMAG2_PG0)	32	RW	FFFF_FFFF
A0C	Secure Memory Access Group Registers (P0SMAG1_PG0)	32	RW	FFFF_FFFF
A10	Partition 1 SDID register (P1SDID_PG0)	32	RO	0000_0000
A14	Secure Memory Access Permissions register (P1SMAPR_PG0)	32	RW	0000_00FF
A18	Secure Memory Access Group Registers (P1SMAG2_PG0)	32	RW	FFFF_FFFF
A1C	Secure Memory Access Group Registers (P1SMAG1_PG0)	32	RW	FFFF_FFFF
A20	Partition 2 SDID register (P2SDID_PG0)	32	RO	0000_0000
A24	Secure Memory Access Permissions register (P2SMAPR_PG0)	32	RW	0000_00FF
A28	Secure Memory Access Group Registers (P2SMAG2_PG0)	32	RW	FFFF_FFFF
A2C	Secure Memory Access Group Registers (P2SMAG1_PG0)	32	RW	FFFF_FFFF
A30	Partition 3 SDID register (P3SDID_PG0)	32	RO	0000_0000
A34	Secure Memory Access Permissions register (P3SMAPR_PG0)	32	RW	0000_00FF
A38	Secure Memory Access Group Registers (P3SMAG2_PG0)	32	RW	FFFF_FFFF
A3C	Secure Memory Access Group Registers (P3SMAG1_PG0)	32	RW	FFFF_FFFF
A40	Partition 4 SDID register (P4SDID_PG0)	32	RO	0000_0000
A44	Secure Memory Access Permissions register (P4SMAPR_PG0)	32	RW	0000_00FF
A48	Secure Memory Access Group Registers (P4SMAG2_PG0)	32	RW	FFFF_FFFF
A4C	Secure Memory Access Group Registers (P4SMAG1_PG0)	32	RW	FFFF_FFFF
A50	Partition 5 SDID register (P5SDID_PG0)	32	RO	0000_0000
A54	Secure Memory Access Permissions register (P5SMAPR_PG0)	32	RW	0000_00FF
A58	Secure Memory Access Group Registers (P5SMAG2_PG0)	32	RW	FFFF_FFFF
A5C	Secure Memory Access Group Registers (P5SMAG1_PG0)	32	RW	FFFF_FFFF
A60	Partition 6 SDID register (P6SDID_PG0)	32	RO	0000_0000
A64	Secure Memory Access Permissions register (P6SMAPR_PG0)	32	RW	0000_00FF
A68	Secure Memory Access Group Registers (P6SMAG2_PG0)	32	RW	FFFF_FFFF
A6C	Secure Memory Access Group Registers (P6SMAG1_PG0)	32	RW	FFFF_FFFF
A70	Partition 7 SDID register (P7SDID_PG0)	32	RO	0000_0000
A74	Secure Memory Access Permissions register (P7SMAPR_PG0)	32	RW	0000_00FF
A78	Secure Memory Access Group Registers (P7SMAG2_PG0)	32	RW	FFFF_FFFF
A7C	Secure Memory Access Group Registers (P7SMAG1_PG0)	32	RW	FFFF_FFFF
B00	Recoverable Error Indication Status (REIS)	32	W1C	0000_0000
B0C	Recoverable Error Indication Halt (REIH)	32	RW	0000_0000
BD0 - BD8	Secure Memory Write Protect Job Ring Register (SMWPJR0R - SMWPJR2R)	32	RW	0000_0000
BE4	Secure Memory Command Register (SMCR_PG0)	32	WO	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
BEC	Secure Memory Command Status Register (SMCSR_PG0)	32	RO	0000_0000
BF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
BFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000
C00	Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR)	64	RO	0000_0000_0000_0000
C08	Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)	64	RO	0000_0000_0000_0000
C10	Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS)	32	RO	0000_0000
C14	Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)	32	RO	0000_0000
C1C	Holding Tank Status (HT0_STATUS)	32	RO	0000_0000
C24	Job Queue Debug Select Register (JQ_DEBUG_SEL)	32	RW	0000_0000
DBC	Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS)	32	RO	0000_0000
DC0	Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)	32	RO	0000_0000
DC4	Job Ring Job-Done Job ID FIFO (JRJDJIF)	32	RO	0000_0000
DE4	Job Ring Job-Done Source 1 (JRJDS1)	32	RO	0000_0000
E00	Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)	64	RO	0000_0000_0000_0000
FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
FBC (alias)	Secure Memory Partition Owners Register (SMPO)	32	RO	5555_0003
FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000
FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004
FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
1000	Input Ring Base Address Register for Job Ring 0 (IRBAR_JR0)	64	RW	0000_0000_0000_0000
100C	Input Ring Size Register for Job Ring 0 (IRSR_JR0)	32	RW	0000_0000
1014	Input Ring Slots Available Register for Job Ring 0 (IRSAR_JR0)	32	RW	0000_0000
101C	Input Ring Jobs Added Register for Job Ring0 (IRJAR_JR0)	32	RW	0000_0000
1020	Output Ring Base Address Register for Job Ring 0 (ORBAR_JR0)	64	RW	0000_0000_0000_0000
102C	Output Ring Size Register for Job Ring 0 (ORSR_JR0)	32	RW	0000_0000
1034	Output Ring Jobs Removed Register for Job Ring 0 (ORJRR_JR0)	32	RW	0000_0000
103C	Output Ring Slots Full Register for Job Ring 0 (ORSFR_JR0)	32	RW	0000_0000
1044	Job Ring Output Status Register for Job Ring 0 (JRSTAR_JR0)	32	RO	0000_0000
104C	Job Ring Interrupt Status Register for Job Ring 0 (JRINTR_JR0)	32	W1C	0000_0000
1050	Job Ring Configuration Register for Job Ring 0, most-significant half (JRCFGR_JR0_MS)	32	RW	0000_0000
1054	Job Ring Configuration Register for Job Ring 0, least-significant half (JRCFGR_JR0_LS)	32	RW	0000_0000
105C	Input Ring Read Index Register for Job Ring 0 (IRRIR_JR0)	32	RW	0000_0000
1064	Output Ring Write Index Register for Job Ring 0 (ORWIR_JR0)	32	RW	0000_0000
106C	Job Ring Command Register for Job Ring 0 (JRCR_JR0)	32	WO	0000_0000
1704	Job Ring 0 Address-Array Valid Register (JR0AAV)	32	RO	0000_0000
1800	Job Ring 0 Address-Array Address 0 Register (JR0AAA0)	64	RO	0000_0000_0000_0000
1808	Job Ring 0 Address-Array Address 1 Register (JR0AAA1)	64	RO	0000_0000_0000_0000
1810	Job Ring 0 Address-Array Address 2 Register (JR0AAA2)	64	RO	0000_0000_0000_0000
1818	Job Ring 0 Address-Array Address 3 Register (JR0AAA3)	64	RO	0000_0000_0000_0000
1A00	Partition 0 SDID register (P0SDID_JR0)	32	RO	0000_0000
1A04	Secure Memory Access Permissions register (P0SMAPR_JR0)	32	RW	0000_00FF
1A08	Secure Memory Access Group Registers (P0SMAG2_JR0)	32	RW	FFFF_FFFF
1A0C	Secure Memory Access Group Registers (P0SMAG1_JR0)	32	RW	FFFF_FFFF
1A10	Partition 1 SDID register (P1SDID_JR0)	32	RO	0000_0000
1A14	Secure Memory Access Permissions register (P1SMAPR_JR0)	32	RW	0000_00FF
1A18	Secure Memory Access Group Registers (P1SMAG2_JR0)	32	RW	FFFF_FFFF
1A1C	Secure Memory Access Group Registers (P1SMAG1_JR0)	32	RW	FFFF_FFFF
1A20	Partition 2 SDID register (P2SDID_JR0)	32	RO	0000_0000
1A24	Secure Memory Access Permissions register (P2SMAPR_JR0)	32	RW	0000_00FF
1A28	Secure Memory Access Group Registers (P2SMAG2_JR0)	32	RW	FFFF_FFFF
1A2C	Secure Memory Access Group Registers (P2SMAG1_JR0)	32	RW	FFFF_FFFF
1A30	Partition 3 SDID register (P3SDID_JR0)	32	RO	0000_0000
1A34	Secure Memory Access Permissions register (P3SMAPR_JR0)	32	RW	0000_00FF

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
1A38	Secure Memory Access Group Registers (P3SMAG2_JR0)	32	RW	FFFF_FFFF
1A3C	Secure Memory Access Group Registers (P3SMAG1_JR0)	32	RW	FFFF_FFFF
1A40	Partition 4 SDID register (P4SDID_JR0)	32	RO	0000_0000
1A44	Secure Memory Access Permissions register (P4SMAPR_JR0)	32	RW	0000_00FF
1A48	Secure Memory Access Group Registers (P4SMAG2_JR0)	32	RW	FFFF_FFFF
1A4C	Secure Memory Access Group Registers (P4SMAG1_JR0)	32	RW	FFFF_FFFF
1A50	Partition 5 SDID register (P5SDID_JR0)	32	RO	0000_0000
1A54	Secure Memory Access Permissions register (P5SMAPR_JR0)	32	RW	0000_00FF
1A58	Secure Memory Access Group Registers (P5SMAG2_JR0)	32	RW	FFFF_FFFF
1A5C	Secure Memory Access Group Registers (P5SMAG1_JR0)	32	RW	FFFF_FFFF
1A60	Partition 6 SDID register (P6SDID_JR0)	32	RO	0000_0000
1A64	Secure Memory Access Permissions register (P6SMAPR_JR0)	32	RW	0000_00FF
1A68	Secure Memory Access Group Registers (P6SMAG2_JR0)	32	RW	FFFF_FFFF
1A6C	Secure Memory Access Group Registers (P6SMAG1_JR0)	32	RW	FFFF_FFFF
1A70	Partition 7 SDID register (P7SDID_JR0)	32	RO	0000_0000
1A74	Secure Memory Access Permissions register (P7SMAPR_JR0)	32	RW	0000_00FF
1A78	Secure Memory Access Group Registers (P7SMAG2_JR0)	32	RW	FFFF_FFFF
1A7C	Secure Memory Access Group Registers (P7SMAG1_JR0)	32	RW	FFFF_FFFF
1BE4	Secure Memory Command Register (SMCR_JR0)	32	WO	0000_0000
1BEC	Secure Memory Command Status Register (SMCSR_JR0)	32	RO	0000_0000
1E00	Recoverable Error Indication Record 0 for Job Ring 0 (REIR0JR0)	32	RO	0000_0000
1E08	Recoverable Error Indication Record 2 for Job Ring 0 (REIR2JR0)	64	RO	0000_0000_0000_0000
1E10	Recoverable Error Indication Record 4 for Job Ring 0 (REIR4JR0)	32	RO	0000_0000
1E14	Recoverable Error Indication Record 5 for Job Ring 0 (REIR5JR0)	32	RO	0000_0000
1FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
1FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
1FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
1FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
1FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
1FBC (alias)	Secure Memory Partition Owners Register (SMPO)	32	RO	5555_0003
1FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000
1FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
1FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
1FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
1FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
1FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
1FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
1FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
1FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
1FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
1FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
1FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
1FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
1FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000
2000	Input Ring Base Address Register for Job Ring 1 (IRBAR_JR1)	64	RW	0000_0000_0000_0000
200C	Input Ring Size Register for Job Ring 1 (IRSR_JR1)	32	RW	0000_0000
2014	Input Ring Slots Available Register for Job Ring 1 (IRSAR_JR1)	32	RW	0000_0000
201C	Input Ring Jobs Added Register for Job Ring1 (IRJAR_JR1)	32	RW	0000_0000
2020	Output Ring Base Address Register for Job Ring 1 (ORBAR_JR1)	64	RW	0000_0000_0000_0000
202C	Output Ring Size Register for Job Ring 1 (ORSR_JR1)	32	RW	0000_0000
2034	Output Ring Jobs Removed Register for Job Ring 1 (ORJRR_JR1)	32	RW	0000_0000
203C	Output Ring Slots Full Register for Job Ring 1 (ORSFR_JR1)	32	RW	0000_0000
2044	Job Ring Output Status Register for Job Ring 1 (JRSTAR_JR1)	32	RO	0000_0000
204C	Job Ring Interrupt Status Register for Job Ring 1 (JRINTR_JR1)	32	W1C	0000_0000
2050	Job Ring Configuration Register for Job Ring 1, most-significant half (JRCFGR_JR1_MS)	32	RW	0000_0000
2054	Job Ring Configuration Register for Job Ring 1, least-significant half (JRCFGR_JR1_LS)	32	RW	0000_0000
205C	Input Ring Read Index Register for Job Ring 1 (IRRIR_JR1)	32	RW	0000_0000
2064	Output Ring Write Index Register for Job Ring 1 (ORWIR_JR1)	32	RW	0000_0000
206C	Job Ring Command Register for Job Ring 1 (JRCR_JR1)	32	WO	0000_0000
2704	Job Ring 1 Address-Array Valid Register (JR1AAV)	32	RO	0000_0000
2800	Job Ring 1 Address-Array Address 0 Register (JR1AAA0)	64	RO	0000_0000_0000_0000
2808	Job Ring 1 Address-Array Address 1 Register (JR1AAA1)	64	RO	0000_0000_0000_0000
2810	Job Ring 1 Address-Array Address 2 Register (JR1AAA2)	64	RO	0000_0000_0000_0000
2818	Job Ring 1 Address-Array Address 3 Register (JR1AAA3)	64	RO	0000_0000_0000_0000
2A00	Partition 0 SDID register (P0SDID_JR1)	32	RO	0000_0000
2A04	Secure Memory Access Permissions register (P0SMAPR_JR1)	32	RW	0000_00FF
2A08	Secure Memory Access Group Registers (P0SMAG2_JR1)	32	RW	FFFF_FFFF
2A0C	Secure Memory Access Group Registers (P0SMAG1_JR1)	32	RW	FFFF_FFFF
2A10	Partition 1 SDID register (P1SDID_JR1)	32	RO	0000_0000
2A14	Secure Memory Access Permissions register (P1SMAPR_JR1)	32	RW	0000_00FF
2A18	Secure Memory Access Group Registers (P1SMAG2_JR1)	32	RW	FFFF_FFFF

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
2A1C	Secure Memory Access Group Registers (P1SMAG1_JR1)	32	RW	FFFF_FFFF
2A20	Partition 2 SDID register (P2SDID_JR1)	32	RO	0000_0000
2A24	Secure Memory Access Permissions register (P2SMAPR_JR1)	32	RW	0000_00FF
2A28	Secure Memory Access Group Registers (P2SMAG2_JR1)	32	RW	FFFF_FFFF
2A2C	Secure Memory Access Group Registers (P2SMAG1_JR1)	32	RW	FFFF_FFFF
2A30	Partition 3 SDID register (P3SDID_JR1)	32	RO	0000_0000
2A34	Secure Memory Access Permissions register (P3SMAPR_JR1)	32	RW	0000_00FF
2A38	Secure Memory Access Group Registers (P3SMAG2_JR1)	32	RW	FFFF_FFFF
2A3C	Secure Memory Access Group Registers (P3SMAG1_JR1)	32	RW	FFFF_FFFF
2A40	Partition 4 SDID register (P4SDID_JR1)	32	RO	0000_0000
2A44	Secure Memory Access Permissions register (P4SMAPR_JR1)	32	RW	0000_00FF
2A48	Secure Memory Access Group Registers (P4SMAG2_JR1)	32	RW	FFFF_FFFF
2A4C	Secure Memory Access Group Registers (P4SMAG1_JR1)	32	RW	FFFF_FFFF
2A50	Partition 5 SDID register (P5SDID_JR1)	32	RO	0000_0000
2A54	Secure Memory Access Permissions register (P5SMAPR_JR1)	32	RW	0000_00FF
2A58	Secure Memory Access Group Registers (P5SMAG2_JR1)	32	RW	FFFF_FFFF
2A5C	Secure Memory Access Group Registers (P5SMAG1_JR1)	32	RW	FFFF_FFFF
2A60	Partition 6 SDID register (P6SDID_JR1)	32	RO	0000_0000
2A64	Secure Memory Access Permissions register (P6SMAPR_JR1)	32	RW	0000_00FF
2A68	Secure Memory Access Group Registers (P6SMAG2_JR1)	32	RW	FFFF_FFFF
2A6C	Secure Memory Access Group Registers (P6SMAG1_JR1)	32	RW	FFFF_FFFF
2A70	Partition 7 SDID register (P7SDID_JR1)	32	RO	0000_0000
2A74	Secure Memory Access Permissions register (P7SMAPR_JR1)	32	RW	0000_00FF
2A78	Secure Memory Access Group Registers (P7SMAG2_JR1)	32	RW	FFFF_FFFF
2A7C	Secure Memory Access Group Registers (P7SMAG1_JR1)	32	RW	FFFF_FFFF
2BE4	Secure Memory Command Register (SMCR_JR1)	32	WO	0000_0000
2BEC	Secure Memory Command Status Register (SMCSR_JR1)	32	RO	0000_0000
2E00	Recoverable Error Indication Record 0 for Job Ring 1 (REIR0JR1)	32	RO	0000_0000
2E08	Recoverable Error Indication Record 2 for Job Ring 1 (REIR2JR1)	64	RO	0000_0000_0000_0000
2E10	Recoverable Error Indication Record 4 for Job Ring 1 (REIR4JR1)	32	RO	0000_0000
2E14	Recoverable Error Indication Record 5 for Job Ring 1 (REIR5JR1)	32	RO	0000_0000
2FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
2FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
2FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
2FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
2FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
2FBC (alias)	Secure Memory Partition Owners Register (SMPO)	32	RO	5555_0003
2FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
2FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
2FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
2FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
2FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
2FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
2FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004
2FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
2FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
2FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
2FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
2FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
2FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
2FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000
3000	Input Ring Base Address Register for Job Ring 2 (IRBAR_JR2)	64	RW	0000_0000_0000_0000
300C	Input Ring Size Register for Job Ring 2 (IRSR_JR2)	32	RW	0000_0000
3014	Input Ring Slots Available Register for Job Ring 2 (IRSAR_JR2)	32	RW	0000_0000
301C	Input Ring Jobs Added Register for Job Ring2 (IRJAR_JR2)	32	RW	0000_0000
3020	Output Ring Base Address Register for Job Ring 2 (ORBAR_JR2)	64	RW	0000_0000_0000_0000
302C	Output Ring Size Register for Job Ring 2 (ORSR_JR2)	32	RW	0000_0000
3034	Output Ring Jobs Removed Register for Job Ring 2 (ORJRR_JR2)	32	RW	0000_0000
303C	Output Ring Slots Full Register for Job Ring 2 (ORSFR_JR2)	32	RW	0000_0000
3044	Job Ring Output Status Register for Job Ring 2 (JRSTAR_JR2)	32	RO	0000_0000
304C	Job Ring Interrupt Status Register for Job Ring 2 (JRINTR_JR2)	32	W1C	0000_0000
3050	Job Ring Configuration Register for Job Ring 2, most-significant half (JRCFGR_JR2_MS)	32	RW	0000_0000
3054	Job Ring Configuration Register for Job Ring 2, least-significant half (JRCFGR_JR2_LS)	32	RW	0000_0000
305C	Input Ring Read Index Register for Job Ring 2 (IRRIR_JR2)	32	RW	0000_0000
3064	Output Ring Write Index Register for Job Ring 2 (ORWIR_JR2)	32	RW	0000_0000
306C	Job Ring Command Register for Job Ring 2 (JRCR_JR2)	32	WO	0000_0000
3704	Job Ring 2 Address-Array Valid Register (JR2AAV)	32	RO	0000_0000
3800	Job Ring 2 Address-Array Address 0 Register (JR2AAA0)	64	RO	0000_0000_0000_0000
3808	Job Ring 2 Address-Array Address 1 Register (JR2AAA1)	64	RO	0000_0000_0000_0000
3810	Job Ring 2 Address-Array Address 2 Register (JR2AAA2)	64	RO	0000_0000_0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
3818	Job Ring 2 Address-Array Address 3 Register (JR2AAA3)	64	RO	0000_0000_0000_0000
3A00	Partition 0 SDID register (P0SDID_JR2)	32	RO	0000_0000
3A04	Secure Memory Access Permissions register (P0SMAPR_JR2)	32	RW	0000_00FF
3A08	Secure Memory Access Group Registers (P0SMAG2_JR2)	32	RW	FFFF_FFFF
3A0C	Secure Memory Access Group Registers (P0SMAG1_JR2)	32	RW	FFFF_FFFF
3A10	Partition 1 SDID register (P1SDID_JR2)	32	RO	0000_0000
3A14	Secure Memory Access Permissions register (P1SMAPR_JR2)	32	RW	0000_00FF
3A18	Secure Memory Access Group Registers (P1SMAG2_JR2)	32	RW	FFFF_FFFF
3A1C	Secure Memory Access Group Registers (P1SMAG1_JR2)	32	RW	FFFF_FFFF
3A20	Partition 2 SDID register (P2SDID_JR2)	32	RO	0000_0000
3A24	Secure Memory Access Permissions register (P2SMAPR_JR2)	32	RW	0000_00FF
3A28	Secure Memory Access Group Registers (P2SMAG2_JR2)	32	RW	FFFF_FFFF
3A2C	Secure Memory Access Group Registers (P2SMAG1_JR2)	32	RW	FFFF_FFFF
3A30	Partition 3 SDID register (P3SDID_JR2)	32	RO	0000_0000
3A34	Secure Memory Access Permissions register (P3SMAPR_JR2)	32	RW	0000_00FF
3A38	Secure Memory Access Group Registers (P3SMAG2_JR2)	32	RW	FFFF_FFFF
3A3C	Secure Memory Access Group Registers (P3SMAG1_JR2)	32	RW	FFFF_FFFF
3A40	Partition 4 SDID register (P4SDID_JR2)	32	RO	0000_0000
3A44	Secure Memory Access Permissions register (P4SMAPR_JR2)	32	RW	0000_00FF
3A48	Secure Memory Access Group Registers (P4SMAG2_JR2)	32	RW	FFFF_FFFF
3A4C	Secure Memory Access Group Registers (P4SMAG1_JR2)	32	RW	FFFF_FFFF
3A50	Partition 5 SDID register (P5SDID_JR2)	32	RO	0000_0000
3A54	Secure Memory Access Permissions register (P5SMAPR_JR2)	32	RW	0000_00FF
3A58	Secure Memory Access Group Registers (P5SMAG2_JR2)	32	RW	FFFF_FFFF
3A5C	Secure Memory Access Group Registers (P5SMAG1_JR2)	32	RW	FFFF_FFFF
3A60	Partition 6 SDID register (P6SDID_JR2)	32	RO	0000_0000
3A64	Secure Memory Access Permissions register (P6SMAPR_JR2)	32	RW	0000_00FF
3A68	Secure Memory Access Group Registers (P6SMAG2_JR2)	32	RW	FFFF_FFFF
3A6C	Secure Memory Access Group Registers (P6SMAG1_JR2)	32	RW	FFFF_FFFF
3A70	Partition 7 SDID register (P7SDID_JR2)	32	RO	0000_0000
3A74	Secure Memory Access Permissions register (P7SMAPR_JR2)	32	RW	0000_00FF
3A78	Secure Memory Access Group Registers (P7SMAG2_JR2)	32	RW	FFFF_FFFF
3A7C	Secure Memory Access Group Registers (P7SMAG1_JR2)	32	RW	FFFF_FFFF
3BE4	Secure Memory Command Register (SMCR_JR2)	32	WO	0000_0000
3BEC	Secure Memory Command Status Register (SMCSR_JR2)	32	RO	0000_0000
3E00	Recoverable Error Indication Record 0 for Job Ring 2 (REIR0JR2)	32	RO	0000_0000
3E08	Recoverable Error Indication Record 2 for Job Ring 2 (REIR2JR2)	64	RO	0000_0000_0000_0000
3E10	Recoverable Error Indication Record 4 for Job Ring 2 (REIR4JR2)	32	RO	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
3E14	Recoverable Error Indication Record 5 for Job Ring 2 (REIR5JR2)	32	RO	0000_0000
3FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
3FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
3FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
3FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
3FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
3FBC (alias)	Secure Memory Partition Owners Register (SMPO)	32	RO	5555_0003
3FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000
3FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
3FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
3FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
3FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
3FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
3FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004
3FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
3FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
3FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
3FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
3FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
3FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
3FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000
6004	RTIC Status Register (RSTA)	32	RO	0004_0000
600C	RTIC Command Register (RCMD)	32	RW	0000_0000
6014	RTIC Control Register (RCTL)	32	RW	0000_0000
601C	RTIC Throttle Register (RTHR)	32	RW	0000_0000
6028	RTIC Watchdog Timer (RWDOG)	64	RW	0000_0000_0000_0000
6034	RTIC Endian Register (REND)	32	RW	0000_0000
6100	RTIC Memory Block A Address 0 Register (RMAA0)	64	RW	0000_0000_0000_0000
610C	RTIC Memory Block A Length 0 Register (RMAL0)	32	RW	0000_0000
6110	RTIC Memory Block A Address 1 Register (RMAA1)	64	RW	0000_0000_0000_0000
611C	RTIC Memory Block A Length 1 Register (RMAL1)	32	RW	0000_0000
6120	RTIC Memory Block B Address 0 Register (RMBA0)	64	RW	0000_0000_0000_0000
612C	RTIC Memory Block B Length 0 Register (RMBL0)	32	RW	0000_0000
6130	RTIC Memory Block B Address 1 Register (RMBA1)	64	RW	0000_0000_0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
613C	RTIC Memory Block B Length 1 Register (RMBL1)	32	RW	0000_0000
6140	RTIC Memory Block C Address 0 Register (RMCA0)	64	RW	0000_0000_0000_0000
614C	RTIC Memory Block C Length 0 Register (RMCL0)	32	RW	0000_0000
6150	RTIC Memory Block C Address 1 Register (RMCA1)	64	RW	0000_0000_0000_0000
615C	RTIC Memory Block C Length 1 Register (RMCL1)	32	RW	0000_0000
6160	RTIC Memory Block D Address 0 Register (RMDA0)	64	RW	0000_0000_0000_0000
616C	RTIC Memory Block D Length 0 Register (RMDL0)	32	RW	0000_0000
6170	RTIC Memory Block D Address 1 Register (RMDA1)	64	RW	0000_0000_0000_0000
617C	RTIC Memory Block D Length 1 Register (RMDL1)	32	RW	0000_0000
6200	RTIC Memory Block A Big Endian Hash Result Word 0 (RAMDB_0)	32	RW	0000_0000
6204	RTIC Memory Block A Big Endian Hash Result Word 1 (RAMDB_1)	32	RW	0000_0000
6208	RTIC Memory Block A Big Endian Hash Result Word 2 (RAMDB_2)	32	RW	0000_0000
620C	RTIC Memory Block A Big Endian Hash Result Word 3 (RAMDB_3)	32	RW	0000_0000
6210	RTIC Memory Block A Big Endian Hash Result Word 4 (RAMDB_4)	32	RW	0000_0000
6214	RTIC Memory Block A Big Endian Hash Result Word 5 (RAMDB_5)	32	RW	0000_0000
6218	RTIC Memory Block A Big Endian Hash Result Word 6 (RAMDB_6)	32	RW	0000_0000
621C	RTIC Memory Block A Big Endian Hash Result Word 7 (RAMDB_7)	32	RW	0000_0000
6220	RTIC Memory Block A Big Endian Hash Result Word 8 (RAMDB_8)	32	RW	0000_0000
6224	RTIC Memory Block A Big Endian Hash Result Word 9 (RAMDB_9)	32	RW	0000_0000
6228	RTIC Memory Block A Big Endian Hash Result Word 10 (RAMDB_10)	32	RW	0000_0000
622C	RTIC Memory Block A Big Endian Hash Result Word 11 (RAMDB_11)	32	RW	0000_0000
6230	RTIC Memory Block A Big Endian Hash Result Word 12 (RAMDB_12)	32	RW	0000_0000
6234	RTIC Memory Block A Big Endian Hash Result Word 13 (RAMDB_13)	32	RW	0000_0000
6238	RTIC Memory Block A Big Endian Hash Result Word 14 (RAMDB_14)	32	RW	0000_0000
623C	RTIC Memory Block A Big Endian Hash Result Word 15 (RAMDB_15)	32	RW	0000_0000
6240	RTIC Memory Block A Big Endian Hash Result Word 16 (RAMDB_16)	32	RW	0000_0000
6244	RTIC Memory Block A Big Endian Hash Result Word 17 (RAMDB_17)	32	RW	0000_0000
6248	RTIC Memory Block A Big Endian Hash Result Word 18 (RAMDB_18)	32	RW	0000_0000
624C	RTIC Memory Block A Big Endian Hash Result Word 19 (RAMDB_19)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
6250	RTIC Memory Block A Big Endian Hash Result Word 20 (RAMDB_20)	32	RW	0000_0000
6254	RTIC Memory Block A Big Endian Hash Result Word 21 (RAMDB_21)	32	RW	0000_0000
6258	RTIC Memory Block A Big Endian Hash Result Word 22 (RAMDB_22)	32	RW	0000_0000
625C	RTIC Memory Block A Big Endian Hash Result Word 23 (RAMDB_23)	32	RW	0000_0000
6260	RTIC Memory Block A Big Endian Hash Result Word 24 (RAMDB_24)	32	RW	0000_0000
6264	RTIC Memory Block A Big Endian Hash Result Word 25 (RAMDB_25)	32	RW	0000_0000
6268	RTIC Memory Block A Big Endian Hash Result Word 26 (RAMDB_26)	32	RW	0000_0000
626C	RTIC Memory Block A Big Endian Hash Result Word 27 (RAMDB_27)	32	RW	0000_0000
6270	RTIC Memory Block A Big Endian Hash Result Word 28 (RAMDB_28)	32	RW	0000_0000
6274	RTIC Memory Block A Big Endian Hash Result Word 29 (RAMDB_29)	32	RW	0000_0000
6278	RTIC Memory Block A Big Endian Hash Result Word 30 (RAMDB_30)	32	RW	0000_0000
627C	RTIC Memory Block A Big Endian Hash Result Word 31 (RAMDB_31)	32	RW	0000_0000
6280	RTIC Memory Block A Little Endian Hash Result Word 0 (RAMDL_0)	32	RW	0000_0000
6284	RTIC Memory Block A Little Endian Hash Result Word 1 (RAMDL_1)	32	RW	0000_0000
6288	RTIC Memory Block A Little Endian Hash Result Word 2 (RAMDL_2)	32	RW	0000_0000
628C	RTIC Memory Block A Little Endian Hash Result Word 3 (RAMDL_3)	32	RW	0000_0000
6290	RTIC Memory Block A Little Endian Hash Result Word 4 (RAMDL_4)	32	RW	0000_0000
6294	RTIC Memory Block A Little Endian Hash Result Word 5 (RAMDL_5)	32	RW	0000_0000
6298	RTIC Memory Block A Little Endian Hash Result Word 6 (RAMDL_6)	32	RW	0000_0000
629C	RTIC Memory Block A Little Endian Hash Result Word 7 (RAMDL_7)	32	RW	0000_0000
62A0	RTIC Memory Block A Little Endian Hash Result Word 8 (RAMDL_8)	32	RW	0000_0000
62A4	RTIC Memory Block A Little Endian Hash Result Word 9 (RAMDL_9)	32	RW	0000_0000
62A8	RTIC Memory Block A Little Endian Hash Result Word 10 (RAMDL_10)	32	RW	0000_0000
62AC	RTIC Memory Block A Little Endian Hash Result Word 11 (RAMDL_11)	32	RW	0000_0000
62B0	RTIC Memory Block A Little Endian Hash Result Word 12 (RAMDL_12)	32	RW	0000_0000
62B4	RTIC Memory Block A Little Endian Hash Result Word 13 (RAMDL_13)	32	RW	0000_0000
62B8	RTIC Memory Block A Little Endian Hash Result Word 14 (RAMDL_14)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
62BC	RTIC Memory Block A Little Endian Hash Result Word 15 (RAMDL_15)	32	RW	0000_0000
62C0	RTIC Memory Block A Little Endian Hash Result Word 16 (RAMDL_16)	32	RW	0000_0000
62C4	RTIC Memory Block A Little Endian Hash Result Word 17 (RAMDL_17)	32	RW	0000_0000
62C8	RTIC Memory Block A Little Endian Hash Result Word 18 (RAMDL_18)	32	RW	0000_0000
62CC	RTIC Memory Block A Little Endian Hash Result Word 19 (RAMDL_19)	32	RW	0000_0000
62D0	RTIC Memory Block A Little Endian Hash Result Word 20 (RAMDL_20)	32	RW	0000_0000
62D4	RTIC Memory Block A Little Endian Hash Result Word 21 (RAMDL_21)	32	RW	0000_0000
62D8	RTIC Memory Block A Little Endian Hash Result Word 22 (RAMDL_22)	32	RW	0000_0000
62DC	RTIC Memory Block A Little Endian Hash Result Word 23 (RAMDL_23)	32	RW	0000_0000
62E0	RTIC Memory Block A Little Endian Hash Result Word 24 (RAMDL_24)	32	RW	0000_0000
62E4	RTIC Memory Block A Little Endian Hash Result Word 25 (RAMDL_25)	32	RW	0000_0000
62E8	RTIC Memory Block A Little Endian Hash Result Word 26 (RAMDL_26)	32	RW	0000_0000
62EC	RTIC Memory Block A Little Endian Hash Result Word 27 (RAMDL_27)	32	RW	0000_0000
62F0	RTIC Memory Block A Little Endian Hash Result Word 28 (RAMDL_28)	32	RW	0000_0000
62F4	RTIC Memory Block A Little Endian Hash Result Word 29 (RAMDL_29)	32	RW	0000_0000
62F8	RTIC Memory Block A Little Endian Hash Result Word 30 (RAMDL_30)	32	RW	0000_0000
62FC	RTIC Memory Block A Little Endian Hash Result Word 31 (RAMDL_31)	32	RW	0000_0000
6300	RTIC Memory Block B Big Endian Hash Result Word 0 (RBMDB_0)	32	RW	0000_0000
6304	RTIC Memory Block B Big Endian Hash Result Word 1 (RBMDB_1)	32	RW	0000_0000
6308	RTIC Memory Block B Big Endian Hash Result Word 2 (RBMDB_2)	32	RW	0000_0000
630C	RTIC Memory Block B Big Endian Hash Result Word 3 (RBMDB_3)	32	RW	0000_0000
6310	RTIC Memory Block B Big Endian Hash Result Word 4 (RBMDB_4)	32	RW	0000_0000
6314	RTIC Memory Block B Big Endian Hash Result Word 5 (RBMDB_5)	32	RW	0000_0000
6318	RTIC Memory Block B Big Endian Hash Result Word 6 (RBMDB_6)	32	RW	0000_0000
631C	RTIC Memory Block B Big Endian Hash Result Word 7 (RBMDB_7)	32	RW	0000_0000
6320	RTIC Memory Block B Big Endian Hash Result Word 8 (RBMDB_8)	32	RW	0000_0000
6324	RTIC Memory Block B Big Endian Hash Result Word 9 (RBMDB_9)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
6328	RTIC Memory Block B Big Endian Hash Result Word 10 (RBMDB_10)	32	RW	0000_0000
632C	RTIC Memory Block B Big Endian Hash Result Word 11 (RBMDB_11)	32	RW	0000_0000
6330	RTIC Memory Block B Big Endian Hash Result Word 12 (RBMDB_12)	32	RW	0000_0000
6334	RTIC Memory Block B Big Endian Hash Result Word 13 (RBMDB_13)	32	RW	0000_0000
6338	RTIC Memory Block B Big Endian Hash Result Word 14 (RBMDB_14)	32	RW	0000_0000
633C	RTIC Memory Block B Big Endian Hash Result Word 15 (RBMDB_15)	32	RW	0000_0000
6340	RTIC Memory Block B Big Endian Hash Result Word 16 (RBMDB_16)	32	RW	0000_0000
6344	RTIC Memory Block B Big Endian Hash Result Word 17 (RBMDB_17)	32	RW	0000_0000
6348	RTIC Memory Block B Big Endian Hash Result Word 18 (RBMDB_18)	32	RW	0000_0000
634C	RTIC Memory Block B Big Endian Hash Result Word 19 (RBMDB_19)	32	RW	0000_0000
6350	RTIC Memory Block B Big Endian Hash Result Word 20 (RBMDB_20)	32	RW	0000_0000
6354	RTIC Memory Block B Big Endian Hash Result Word 21 (RBMDB_21)	32	RW	0000_0000
6358	RTIC Memory Block B Big Endian Hash Result Word 22 (RBMDB_22)	32	RW	0000_0000
635C	RTIC Memory Block B Big Endian Hash Result Word 23 (RBMDB_23)	32	RW	0000_0000
6360	RTIC Memory Block B Big Endian Hash Result Word 24 (RBMDB_24)	32	RW	0000_0000
6364	RTIC Memory Block B Big Endian Hash Result Word 25 (RBMDB_25)	32	RW	0000_0000
6368	RTIC Memory Block B Big Endian Hash Result Word 26 (RBMDB_26)	32	RW	0000_0000
636C	RTIC Memory Block B Big Endian Hash Result Word 27 (RBMDB_27)	32	RW	0000_0000
6370	RTIC Memory Block B Big Endian Hash Result Word 28 (RBMDB_28)	32	RW	0000_0000
6374	RTIC Memory Block B Big Endian Hash Result Word 29 (RBMDB_29)	32	RW	0000_0000
6378	RTIC Memory Block B Big Endian Hash Result Word 30 (RBMDB_30)	32	RW	0000_0000
637C	RTIC Memory Block B Big Endian Hash Result Word 31 (RBMDB_31)	32	RW	0000_0000
6380	RTIC Memory Block B Little Endian Hash Result Word 0 (RBMDL_0)	32	RW	0000_0000
6384	RTIC Memory Block B Little Endian Hash Result Word 1 (RBMDL_1)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
6388	RTIC Memory Block B Little Endian Hash Result Word 2 (RBMDL_2)	32	RW	0000_0000
638C	RTIC Memory Block B Little Endian Hash Result Word 3 (RBMDL_3)	32	RW	0000_0000
6390	RTIC Memory Block B Little Endian Hash Result Word 4 (RBMDL_4)	32	RW	0000_0000
6394	RTIC Memory Block B Little Endian Hash Result Word 5 (RBMDL_5)	32	RW	0000_0000
6398	RTIC Memory Block B Little Endian Hash Result Word 6 (RBMDL_6)	32	RW	0000_0000
639C	RTIC Memory Block B Little Endian Hash Result Word 7 (RBMDL_7)	32	RW	0000_0000
63A0	RTIC Memory Block B Little Endian Hash Result Word 8 (RBMDL_8)	32	RW	0000_0000
63A4	RTIC Memory Block B Little Endian Hash Result Word 9 (RBMDL_9)	32	RW	0000_0000
63A8	RTIC Memory Block B Little Endian Hash Result Word 10 (RBMDL_10)	32	RW	0000_0000
63AC	RTIC Memory Block B Little Endian Hash Result Word 11 (RBMDL_11)	32	RW	0000_0000
63B0	RTIC Memory Block B Little Endian Hash Result Word 12 (RBMDL_12)	32	RW	0000_0000
63B4	RTIC Memory Block B Little Endian Hash Result Word 13 (RBMDL_13)	32	RW	0000_0000
63B8	RTIC Memory Block B Little Endian Hash Result Word 14 (RBMDL_14)	32	RW	0000_0000
63BC	RTIC Memory Block B Little Endian Hash Result Word 15 (RBMDL_15)	32	RW	0000_0000
63C0	RTIC Memory Block B Little Endian Hash Result Word 16 (RBMDL_16)	32	RW	0000_0000
63C4	RTIC Memory Block B Little Endian Hash Result Word 17 (RBMDL_17)	32	RW	0000_0000
63C8	RTIC Memory Block B Little Endian Hash Result Word 18 (RBMDL_18)	32	RW	0000_0000
63CC	RTIC Memory Block B Little Endian Hash Result Word 19 (RBMDL_19)	32	RW	0000_0000
63D0	RTIC Memory Block B Little Endian Hash Result Word 20 (RBMDL_20)	32	RW	0000_0000
63D4	RTIC Memory Block B Little Endian Hash Result Word 21 (RBMDL_21)	32	RW	0000_0000
63D8	RTIC Memory Block B Little Endian Hash Result Word 22 (RBMDL_22)	32	RW	0000_0000
63DC	RTIC Memory Block B Little Endian Hash Result Word 23 (RBMDL_23)	32	RW	0000_0000
63E0	RTIC Memory Block B Little Endian Hash Result Word 24 (RBMDL_24)	32	RW	0000_0000
63E4	RTIC Memory Block B Little Endian Hash Result Word 25 (RBMDL_25)	32	RW	0000_0000
63E8	RTIC Memory Block B Little Endian Hash Result Word 26 (RBMDL_26)	32	RW	0000_0000
63EC	RTIC Memory Block B Little Endian Hash Result Word 27 (RBMDL_27)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
63F0	RTIC Memory Block B Little Endian Hash Result Word 28 (RBMDL_28)	32	RW	0000_0000
63F4	RTIC Memory Block B Little Endian Hash Result Word 29 (RBMDL_29)	32	RW	0000_0000
63F8	RTIC Memory Block B Little Endian Hash Result Word 30 (RBMDL_30)	32	RW	0000_0000
63FC	RTIC Memory Block B Little Endian Hash Result Word 31 (RBMDL_31)	32	RW	0000_0000
6400	RTIC Memory Block C Big Endian Hash Result Word 0 (RCMDB_0)	32	RW	0000_0000
6404	RTIC Memory Block C Big Endian Hash Result Word 1 (RCMDB_1)	32	RW	0000_0000
6408	RTIC Memory Block C Big Endian Hash Result Word 2 (RCMDB_2)	32	RW	0000_0000
640C	RTIC Memory Block C Big Endian Hash Result Word 3 (RCMDB_3)	32	RW	0000_0000
6410	RTIC Memory Block C Big Endian Hash Result Word 4 (RCMDB_4)	32	RW	0000_0000
6414	RTIC Memory Block C Big Endian Hash Result Word 5 (RCMDB_5)	32	RW	0000_0000
6418	RTIC Memory Block C Big Endian Hash Result Word 6 (RCMDB_6)	32	RW	0000_0000
641C	RTIC Memory Block C Big Endian Hash Result Word 7 (RCMDB_7)	32	RW	0000_0000
6420	RTIC Memory Block C Big Endian Hash Result Word 8 (RCMDB_8)	32	RW	0000_0000
6424	RTIC Memory Block C Big Endian Hash Result Word 9 (RCMDB_9)	32	RW	0000_0000
6428	RTIC Memory Block C Big Endian Hash Result Word 10 (RCMDB_10)	32	RW	0000_0000
642C	RTIC Memory Block C Big Endian Hash Result Word 11 (RCMDB_11)	32	RW	0000_0000
6430	RTIC Memory Block C Big Endian Hash Result Word 12 (RCMDB_12)	32	RW	0000_0000
6434	RTIC Memory Block C Big Endian Hash Result Word 13 (RCMDB_13)	32	RW	0000_0000
6438	RTIC Memory Block C Big Endian Hash Result Word 14 (RCMDB_14)	32	RW	0000_0000
643C	RTIC Memory Block C Big Endian Hash Result Word 15 (RCMDB_15)	32	RW	0000_0000
6440	RTIC Memory Block C Big Endian Hash Result Word 16 (RCMDB_16)	32	RW	0000_0000
6444	RTIC Memory Block C Big Endian Hash Result Word 17 (RCMDB_17)	32	RW	0000_0000
6448	RTIC Memory Block C Big Endian Hash Result Word 18 (RCMDB_18)	32	RW	0000_0000
644C	RTIC Memory Block C Big Endian Hash Result Word 19 (RCMDB_19)	32	RW	0000_0000
6450	RTIC Memory Block C Big Endian Hash Result Word 20 (RCMDB_20)	32	RW	0000_0000
6454	RTIC Memory Block C Big Endian Hash Result Word 21 (RCMDB_21)	32	RW	0000_0000
6458	RTIC Memory Block C Big Endian Hash Result Word 22 (RCMDB_22)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
645C	RTIC Memory Block C Big Endian Hash Result Word 23 (RCMDB_23)	32	RW	0000_0000
6460	RTIC Memory Block C Big Endian Hash Result Word 24 (RCMDB_24)	32	RW	0000_0000
6464	RTIC Memory Block C Big Endian Hash Result Word 25 (RCMDB_25)	32	RW	0000_0000
6468	RTIC Memory Block C Big Endian Hash Result Word 26 (RCMDB_26)	32	RW	0000_0000
646C	RTIC Memory Block C Big Endian Hash Result Word 27 (RCMDB_27)	32	RW	0000_0000
6470	RTIC Memory Block C Big Endian Hash Result Word 28 (RCMDB_28)	32	RW	0000_0000
6474	RTIC Memory Block C Big Endian Hash Result Word 29 (RCMDB_29)	32	RW	0000_0000
6478	RTIC Memory Block C Big Endian Hash Result Word 30 (RCMDB_30)	32	RW	0000_0000
647C	RTIC Memory Block C Big Endian Hash Result Word 31 (RCMDB_31)	32	RW	0000_0000
6480	RTIC Memory Block C Little Endian Hash Result Word 0 (RCMDL_0)	32	RW	0000_0000
6484	RTIC Memory Block C Little Endian Hash Result Word 1 (RCMDL_1)	32	RW	0000_0000
6488	RTIC Memory Block C Little Endian Hash Result Word 2 (RCMDL_2)	32	RW	0000_0000
648C	RTIC Memory Block C Little Endian Hash Result Word 3 (RCMDL_3)	32	RW	0000_0000
6490	RTIC Memory Block C Little Endian Hash Result Word 4 (RCMDL_4)	32	RW	0000_0000
6494	RTIC Memory Block C Little Endian Hash Result Word 5 (RCMDL_5)	32	RW	0000_0000
6498	RTIC Memory Block C Little Endian Hash Result Word 6 (RCMDL_6)	32	RW	0000_0000
649C	RTIC Memory Block C Little Endian Hash Result Word 7 (RCMDL_7)	32	RW	0000_0000
64A0	RTIC Memory Block C Little Endian Hash Result Word 8 (RCMDL_8)	32	RW	0000_0000
64A4	RTIC Memory Block C Little Endian Hash Result Word 9 (RCMDL_9)	32	RW	0000_0000
64A8	RTIC Memory Block C Little Endian Hash Result Word 10 (RCMDL_10)	32	RW	0000_0000
64AC	RTIC Memory Block C Little Endian Hash Result Word 11 (RCMDL_11)	32	RW	0000_0000
64B0	RTIC Memory Block C Little Endian Hash Result Word 12 (RCMDL_12)	32	RW	0000_0000
64B4	RTIC Memory Block C Little Endian Hash Result Word 13 (RCMDL_13)	32	RW	0000_0000
64B8	RTIC Memory Block C Little Endian Hash Result Word 14 (RCMDL_14)	32	RW	0000_0000
64BC	RTIC Memory Block C Little Endian Hash Result Word 15 (RCMDL_15)	32	RW	0000_0000
64C0	RTIC Memory Block C Little Endian Hash Result Word 16 (RCMDL_16)	32	RW	0000_0000
64C4	RTIC Memory Block C Little Endian Hash Result Word 17 (RCMDL_17)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
64C8	RTIC Memory Block C Little Endian Hash Result Word 18 (RCMDL_18)	32	RW	0000_0000
64CC	RTIC Memory Block C Little Endian Hash Result Word 19 (RCMDL_19)	32	RW	0000_0000
64D0	RTIC Memory Block C Little Endian Hash Result Word 20 (RCMDL_20)	32	RW	0000_0000
64D4	RTIC Memory Block C Little Endian Hash Result Word 21 (RCMDL_21)	32	RW	0000_0000
64D8	RTIC Memory Block C Little Endian Hash Result Word 22 (RCMDL_22)	32	RW	0000_0000
64DC	RTIC Memory Block C Little Endian Hash Result Word 23 (RCMDL_23)	32	RW	0000_0000
64E0	RTIC Memory Block C Little Endian Hash Result Word 24 (RCMDL_24)	32	RW	0000_0000
64E4	RTIC Memory Block C Little Endian Hash Result Word 25 (RCMDL_25)	32	RW	0000_0000
64E8	RTIC Memory Block C Little Endian Hash Result Word 26 (RCMDL_26)	32	RW	0000_0000
64EC	RTIC Memory Block C Little Endian Hash Result Word 27 (RCMDL_27)	32	RW	0000_0000
64F0	RTIC Memory Block C Little Endian Hash Result Word 28 (RCMDL_28)	32	RW	0000_0000
64F4	RTIC Memory Block C Little Endian Hash Result Word 29 (RCMDL_29)	32	RW	0000_0000
64F8	RTIC Memory Block C Little Endian Hash Result Word 30 (RCMDL_30)	32	RW	0000_0000
64FC	RTIC Memory Block C Little Endian Hash Result Word 31 (RCMDL_31)	32	RW	0000_0000
6500	RTIC Memory Block D Big Endian Hash Result Word 0 (RDMDB_0)	32	RW	0000_0000
6504	RTIC Memory Block D Big Endian Hash Result Word 1 (RDMDB_1)	32	RW	0000_0000
6508	RTIC Memory Block D Big Endian Hash Result Word 2 (RDMDB_2)	32	RW	0000_0000
650C	RTIC Memory Block D Big Endian Hash Result Word 3 (RDMDB_3)	32	RW	0000_0000
6510	RTIC Memory Block D Big Endian Hash Result Word 4 (RDMDB_4)	32	RW	0000_0000
6514	RTIC Memory Block D Big Endian Hash Result Word 5 (RDMDB_5)	32	RW	0000_0000
6518	RTIC Memory Block D Big Endian Hash Result Word 6 (RDMDB_6)	32	RW	0000_0000
651C	RTIC Memory Block D Big Endian Hash Result Word 7 (RDMDB_7)	32	RW	0000_0000
6520	RTIC Memory Block D Big Endian Hash Result Word 8 (RDMDB_8)	32	RW	0000_0000
6524	RTIC Memory Block D Big Endian Hash Result Word 9 (RDMDB_9)	32	RW	0000_0000
6528	RTIC Memory Block D Big Endian Hash Result Word 10 (RDMDB_10)	32	RW	0000_0000
652C	RTIC Memory Block D Big Endian Hash Result Word 11 (RDMDB_11)	32	RW	0000_0000
6530	RTIC Memory Block D Big Endian Hash Result Word 12 (RDMDB_12)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
6534	RTIC Memory Block D Big Endian Hash Result Word 13 (RDMDB_13)	32	RW	0000_0000
6538	RTIC Memory Block D Big Endian Hash Result Word 14 (RDMDB_14)	32	RW	0000_0000
653C	RTIC Memory Block D Big Endian Hash Result Word 15 (RDMDB_15)	32	RW	0000_0000
6540	RTIC Memory Block D Big Endian Hash Result Word 16 (RDMDB_16)	32	RW	0000_0000
6544	RTIC Memory Block D Big Endian Hash Result Word 17 (RDMDB_17)	32	RW	0000_0000
6548	RTIC Memory Block D Big Endian Hash Result Word 18 (RDMDB_18)	32	RW	0000_0000
654C	RTIC Memory Block D Big Endian Hash Result Word 19 (RDMDB_19)	32	RW	0000_0000
6550	RTIC Memory Block D Big Endian Hash Result Word 20 (RDMDB_20)	32	RW	0000_0000
6554	RTIC Memory Block D Big Endian Hash Result Word 21 (RDMDB_21)	32	RW	0000_0000
6558	RTIC Memory Block D Big Endian Hash Result Word 22 (RDMDB_22)	32	RW	0000_0000
655C	RTIC Memory Block D Big Endian Hash Result Word 23 (RDMDB_23)	32	RW	0000_0000
6560	RTIC Memory Block D Big Endian Hash Result Word 24 (RDMDB_24)	32	RW	0000_0000
6564	RTIC Memory Block D Big Endian Hash Result Word 25 (RDMDB_25)	32	RW	0000_0000
6568	RTIC Memory Block D Big Endian Hash Result Word 26 (RDMDB_26)	32	RW	0000_0000
656C	RTIC Memory Block D Big Endian Hash Result Word 27 (RDMDB_27)	32	RW	0000_0000
6570	RTIC Memory Block D Big Endian Hash Result Word 28 (RDMDB_28)	32	RW	0000_0000
6574	RTIC Memory Block D Big Endian Hash Result Word 29 (RDMDB_29)	32	RW	0000_0000
6578	RTIC Memory Block D Big Endian Hash Result Word 30 (RDMDB_30)	32	RW	0000_0000
657C	RTIC Memory Block D Big Endian Hash Result Word 31 (RDMDB_31)	32	RW	0000_0000
6580	RTIC Memory Block D Little Endian Hash Result Word 0 (RDMDL_0)	32	RW	0000_0000
6584	RTIC Memory Block D Little Endian Hash Result Word 1 (RDMDL_1)	32	RW	0000_0000
6588	RTIC Memory Block D Little Endian Hash Result Word 2 (RDMDL_2)	32	RW	0000_0000
658C	RTIC Memory Block D Little Endian Hash Result Word 3 (RDMDL_3)	32	RW	0000_0000
6590	RTIC Memory Block D Little Endian Hash Result Word 4 (RDMDL_4)	32	RW	0000_0000
6594	RTIC Memory Block D Little Endian Hash Result Word 5 (RDMDL_5)	32	RW	0000_0000
6598	RTIC Memory Block D Little Endian Hash Result Word 6 (RDMDL_6)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
659C	RTIC Memory Block D Little Endian Hash Result Word 7 (RDMDL_7)	32	RW	0000_0000
65A0	RTIC Memory Block D Little Endian Hash Result Word 8 (RDMDL_8)	32	RW	0000_0000
65A4	RTIC Memory Block D Little Endian Hash Result Word 9 (RDMDL_9)	32	RW	0000_0000
65A8	RTIC Memory Block D Little Endian Hash Result Word 10 (RDMDL_10)	32	RW	0000_0000
65AC	RTIC Memory Block D Little Endian Hash Result Word 11 (RDMDL_11)	32	RW	0000_0000
65B0	RTIC Memory Block D Little Endian Hash Result Word 12 (RDMDL_12)	32	RW	0000_0000
65B4	RTIC Memory Block D Little Endian Hash Result Word 13 (RDMDL_13)	32	RW	0000_0000
65B8	RTIC Memory Block D Little Endian Hash Result Word 14 (RDMDL_14)	32	RW	0000_0000
65BC	RTIC Memory Block D Little Endian Hash Result Word 15 (RDMDL_15)	32	RW	0000_0000
65C0	RTIC Memory Block D Little Endian Hash Result Word 16 (RDMDL_16)	32	RW	0000_0000
65C4	RTIC Memory Block D Little Endian Hash Result Word 17 (RDMDL_17)	32	RW	0000_0000
65C8	RTIC Memory Block D Little Endian Hash Result Word 18 (RDMDL_18)	32	RW	0000_0000
65CC	RTIC Memory Block D Little Endian Hash Result Word 19 (RDMDL_19)	32	RW	0000_0000
65D0	RTIC Memory Block D Little Endian Hash Result Word 20 (RDMDL_20)	32	RW	0000_0000
65D4	RTIC Memory Block D Little Endian Hash Result Word 21 (RDMDL_21)	32	RW	0000_0000
65D8	RTIC Memory Block D Little Endian Hash Result Word 22 (RDMDL_22)	32	RW	0000_0000
65DC	RTIC Memory Block D Little Endian Hash Result Word 23 (RDMDL_23)	32	RW	0000_0000
65E0	RTIC Memory Block D Little Endian Hash Result Word 24 (RDMDL_24)	32	RW	0000_0000
65E4	RTIC Memory Block D Little Endian Hash Result Word 25 (RDMDL_25)	32	RW	0000_0000
65E8	RTIC Memory Block D Little Endian Hash Result Word 26 (RDMDL_26)	32	RW	0000_0000
65EC	RTIC Memory Block D Little Endian Hash Result Word 27 (RDMDL_27)	32	RW	0000_0000
65F0	RTIC Memory Block D Little Endian Hash Result Word 28 (RDMDL_28)	32	RW	0000_0000
65F4	RTIC Memory Block D Little Endian Hash Result Word 29 (RDMDL_29)	32	RW	0000_0000
65F8	RTIC Memory Block D Little Endian Hash Result Word 30 (RDMDL_30)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
65FC	RTIC Memory Block D Little Endian Hash Result Word 31 (RDMDL_31)	32	RW	0000_0000
6E00	Recoverable Error Indication Record 0 for RTIC (REIR0RTIC)	32	RO	0000_0000
6E08	Recoverable Error Indication Record 2 for RTIC (REIR2RTIC)	64	RO	0000_0000_0000_0000
6E10	Recoverable Error Indication Record 4 for RTIC (REIR4RTIC)	32	RO	0000_0000
6E14	Recoverable Error Indication Record 5 for RTIC (REIR5RTIC)	32	RO	0000_0000
6FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
6FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
6FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
6FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
6FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
6FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000
6FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
6FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
6FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
6FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
6FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
6FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004
6FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
6FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
6FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
6FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
6FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
6FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
6FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000
8004	CCB 0 Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR)	32	RW	0000_0000
8004	CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)	32	RW	0000_0000
8004	CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG)	32	RW	0000_0000
800C	CCB 0 Class 1 Key Size Register (C0C1KSR)	32	RW	0000_0000
8010	CCB 0 Class 1 Data Size Register (C0C1DSR)	64	RW	0000_0000_0000_0000
801C	CCB 0 Class 1 ICV Size Register (C0C1ICVSR)	32	RW	0000_0000
8034	CCB 0 CHA Control Register (C0CCTRL)	32	WO	0000_0000
803C	CCB 0 Interrupt Control Register (C0ICTL)	32	W1C	0000_0000
8044	CCB 0 Clear Written Register (C0CWR)	32	WO	0000_0000
8048	CCB 0 Status and Error Register, most-significant half (C0CSTA_MS)	32	RO	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
804C	CCB 0 Status and Error Register, least-significant half (C0CSTA_LS)	32	RO	0000_0000
805C	CCB 0 Class 1 AAD Size Register (C0C1AADSZR)	32	RW	0000_0000
8064	CCB 0 Class 1 IV Size Register (C0C1IVSZR)	32	RW	0000_0000
8084	PKHA A Size Register (C0PKASZR)	32	RW	0000_0000
808C	PKHA B Size Register (C0PKBSZR)	32	RW	0000_0000
8094	PKHA N Size Register (C0PKNSZR)	32	RW	0000_0000
809C	PKHA E Size Register (C0PKESZR)	32	RW	0000_0000
8100 - 813C	CCB 0 Class 1 Context Register Word a (C0C1CTXR0 - C0C1CTXR 15)	32	RW	0000_0000
8200 - 821C	CCB 0 Class 1 Key Registers Word a (C0C1KR0 - C0C1KR7)	32	RW	0000_0000
8404	CCB 0 Class 2 Mode Register (C0C2MR)	32	RW	0000_0000
840C	CCB 0 Class 2 Key Size Register (C0C2KSR)	32	RW	0000_0000
8410	CCB 0 Class 2 Data Size Register (C0C2DSR)	64	RW	0000_0000_0000_0000
841C	CCB 0 Class 2 ICV Size Register (C0C2ICVSZR)	32	RW	0000_0000
8500 - 8524	CCB 0 Class 2 Context Register Word a (C0C2CTXR0 - C0C2CTXR 9)	32	RW	0000_0000
8600 - 863C	CCB 0 Class 2 Key Register Word a (C0C2KEYR0 - C0C2KEYR15)	32	RW	0000_0000
87C0	CCB 0 FIFO Status Register (C0FIFOSTA)	32	RO	0000_0000
87D0	CCB 0 iNformation FIFO When STYPE != 10b (C0NFIFO)	32	WO	0000_0000
87D0	CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2)	32	WO	0000_0000
87E0	CCB 0 Input Data FIFO (C0IFIFO)	32	WO	0000_0000
87F0	CCB 0 Output Data FIFO (C0OFIFO)	64	RO	0000_0000_0000_0000
8800	DECO0 Job Queue Control Register, most-significant half (D0JQCR_MS)	32	RW	0000_0000
8804	DECO0 Job Queue Control Register, least-significant half (D0JQCR_LS)	32	RO	0000_0000
8808	DECO0 Descriptor Address Register (D0DAR)	64	RO	0000_0000_0000_0000
8810	DECO0 Operation Status Register, most-significant half (D0OPSTA_MS)	32	RO	0000_0000
8814	DECO0 Operation Status Register, least-significant half (D0OPSTA_LS)	32	RO	0000_0000
8820	DECO0 Primary DID Status Register (D0PDIDSR)	32	RO	0000_0000
8824	DECO0 Output DID Status Register (D0ODIDSR)	32	RO	0000_0000
8840	DECO0 Math Register 0_MS (D0MTH0_MS)	32	RW	0000_0000
8844	DECO0 Math Register 0_LS (D0MTH0_LS)	32	RW	0000_0000
8848	DECO0 Math Register 1_MS (D0MTH1_MS)	32	RW	0000_0000
884C	DECO0 Math Register 1_LS (D0MTH1_LS)	32	RW	0000_0000
8850	DECO0 Math Register 2_MS (D0MTH2_MS)	32	RW	0000_0000
8854	DECO0 Math Register 2_LS (D0MTH2_LS)	32	RW	0000_0000

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
8858	DECO0 Math Register 3_MS (D0MTH3_MS)	32	RW	0000_0000
885C	DECO0 Math Register 3_LS (D0MTH3_LS)	32	RW	0000_0000
8880	DECO0 Gather Table Register 0 Word 0 (D0GTR0_0)	32	RU	0000_0000
8884	DECO0 Gather Table Register 0 Word 1 (D0GTR0_1)	32	RW	0000_0000
8888	DECO0 Gather Table Register 0 Word 2 (D0GTR0_2)	32	RW	0000_0000
888C	DECO0 Gather Table Register 0 Word 3 (D0GTR0_3)	32	RW	0000_0000
8900	DECO0 Scatter Table Register 0 Word 0 (D0STR0_0)	32	RU	0000_0000
8904	DECO0 Scatter Table Register 0 Word 1 (D0STR0_1)	32	RW	0000_0000
8908	DECO0 Scatter Table Register 0 Word 2 (D0STR0_2)	32	RW	0000_0000
890C	DECO0 Scatter Table Register 0 Word 3 (D0STR0_3)	32	RW	0000_0000
8A00 - 8AFC	DECO0 Descriptor Buffer Word a (D0DESB0 - D0DESB63)	32	RW	0000_0000
8E00	DECO0 Debug Job Register (D0DJR)	32	RO	0000_0000
8E04	DECO0 Debug DECO Register (D0DDR)	32	RO	0000_0000
8E08	DECO0 Debug Job Pointer (D0DJP)	64	RO	0000_0000_0000_0000
8E10	DECO0 Debug Shared Pointer (D0SDP)	64	RO	0000_0000_0000_0000
8E18	DECO0 Debug DID, most-significant half (D0DDR_MS)	32	RO	0000_0000
8E1C	DECO0 Debug DID, least-significant half (D0DDR_LS)	32	RO	0000_0000
8E20	Sequence Output Length Register (SOL0)	32	RW	0000_0000
8E24	Variable Sequence Output Length Register (VSOL0)	32	RW	0000_0000
8E28	Sequence Input Length Register (SIL0)	32	RW	0000_0000
8E2C	Variable Sequence Input Length Register (VSIL0)	32	RW	0000_0000
8E30	Protocol Override Register (D0POVRD)	32	RW	0000_0000
8E34	Variable Sequence Output Length Register; Upper 32 bits (UVSOL0)	32	RW	0000_0000
8E38	Variable Sequence Input Length Register; Upper 32 bits (UVSIL0)	32	RW	0000_0000
8FA0 (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	0000_0000
8FA4 (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	6003_411A
8FA8 (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	0191_4201
8FAC (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_6403
8FB4 (alias)	Secure Memory Status Register (SMSTA)	32	RO	0000_0000
8FC0 (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000
8FC8 (alias)	Fault Address DID Register (FADID)	32	RO	0000_0000
8FCC (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000
8FD4 (alias)	CAAM Status Register (CSTA)	32	RO	0000_0002
8FD8 (alias)	Secure Memory Version ID Register, most-significant half (SMVID_MS)	32	RO	000F_7007
8FDC (alias)	Secure Memory Version ID Register, least-significant half (SMVID_LS)	32	RO	0002_0300
8FE0 (alias)	RTIC Version ID Register (RVID)	32	RO	0F02_0004

Table continues on the next page...

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
8FE4 (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0900_0005
8FE8 (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	4500_0000
8FEC (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	1004_0133
8FF0 (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	3100_0000
8FF4 (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1001_1111
8FF8 (alias)	CAAM Version ID Register, most-significant half (CAAMVID_MS)	32	RO	0A16_0401
8FFC (alias)	CAAM Version ID Register, least-significant half (CAAMVID_LS)	32	RO	0000_0000

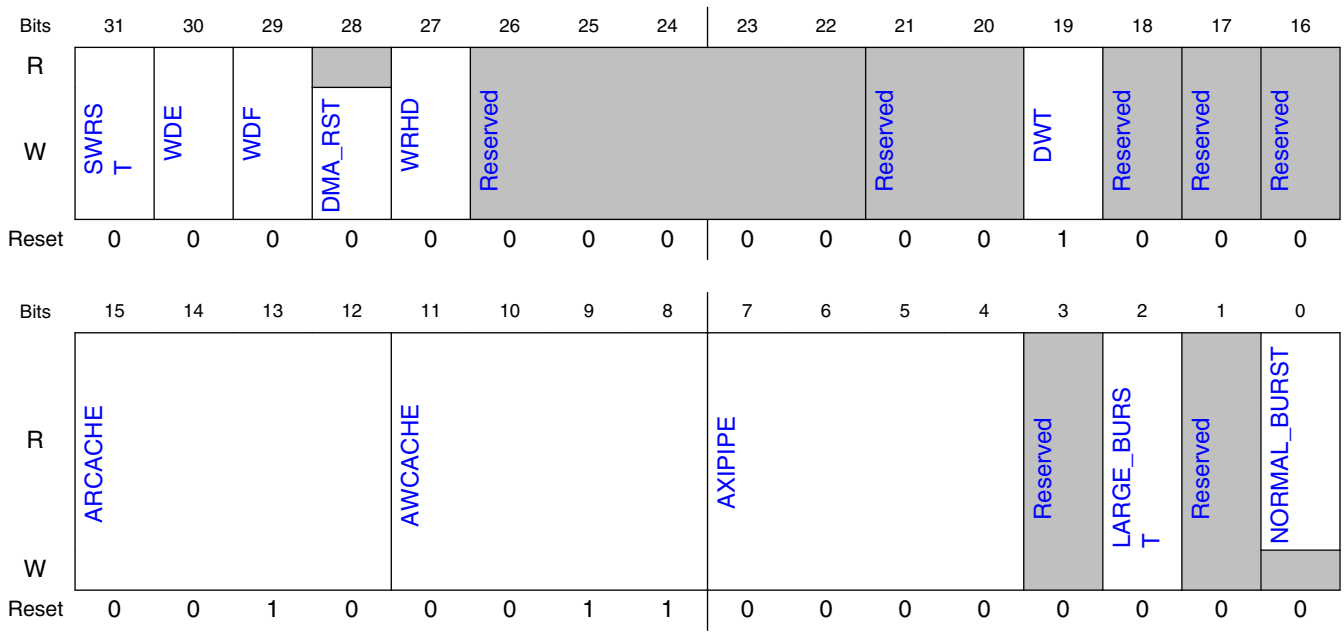
10.13.2 Master Configuration Register (MCFGR)

The Master Configuration Register is used to set some bus master configurations. This register is typically written at boot time, and in some debug scenarios.

10.13.2.1 Offset

Register	Offset
MCFGR	4h

10.13.2.2 Diagram



10.13.2.3 Fields

Field	Description
31 SWRST	Software Reset. Writing a 1 to this bit will cause most registers and state machines in CAAM to reset. The following CAAM registers are not reset: MCFGR, PAGE0_SDID, SCFGR, JR0DID_MS, JR0DID_LS, JR1DID_MS, JR1DID_LS, JR2DID_MS, JR2DID_LS, JRSTARTR, RTIC_OWN, RTICADID_MS, RTICADID_LS, RTICBDID_MS, RTICBDID_LS, RTICCDID_MS, RTICCDID_LS, RTICDDID_MS, RTICDDID_LS, DAR, PBSL, JDKEKR_0 - JDKEKR_7, TDKEKR_0 - TDKEKR_7, TDSKR_0 - TDSKR_7, SKNR, RTMCTL, RTSCMISC, RTPKRRNG, RTPKRMAX, RTPKRSQ, RTSDCTL, RTTOTSAM, RTSBLIM, RTFRQMIN, RTFRQCNT, RTFRQMAX, RTSCML, RTSCMC, RTSCR1L, RTSCR1C, RTSCR2C, RTSCR2L, RTSCR3L, RTSCR3C, RTSCR4L, RTSCR4C, RTSCR5L, RTSCR5C, RTSCR6PC, RTSCR6PL, RTSTATUS, RTENT0 - RTENT11, RTPKRCNT10, RTPKRCNT32, RTPKRCNT54, RTPKRCNT76, RTPKRCNT98, RTPKRCNTBA, RTPKRCNTDC, RTPKRCNTFE, RDSTA, RDINT0, RDINT1, RDHCNTL, RDHDIG, RDHBUF, JR0SMVBA, JR1SMVBA, JR2SMVBA, SMWPJR0R, SMWPJR1R, SMWPJR2R, but the remaining registers in CAAM register page 0 **are** reset by SWRST. The Job Ring registers in CAAM register pages 1 .. 3 **are** reset by SWRST. However the following Secure Memory registers in CAAM register pages 1 .. 3 **are not** reset by SWRST: SMSTA, SMPO, SMCR, SMCSR, PxSDIDR_JRy for Secure Memory Partitions x= 0 .. 7 and JRs y= 0 .. 2, PxSMAPR_JRy, PxSMAG2_JRy, PxSMAG1_JRy, SMCR_JRy and SMCSR_JRy and JRs y= 0 .. 2,The RTIC registers in CAAM register page 6**are not** reset by SWRST. (If an RTIC descriptor is in execution or is waiting for execution when SWRST is requested, RTIC will abandon the current sweep through all the hash blocks and restart hashing at the first hash block.) The DECO and CCB registers in CAAM register page 8 **are** reset by SWRST. 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 LARGE_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.

Table continues on the next page...

Field	Description
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:** The watchdog expiration period is extended for certain DECO operations and for RNG reseeding, because these can take longer than the normal watchdog expiration period.
29 WDF	Watchdog Fast. Causes the DECO Watchdog Timer to expire prematurely for testing purposes. To facilitate testing the upper bytes of the encrypted byte count logic, when WDF is 1 the encrypted byte count increments by 2^{16} per byte that is encrypted. When WDF is 0 the encrypted byte count increments by 1 per byte that is encrypted.
28 DMA_RST	DMA Reset. If SWRST is already 1, writing a 1 to DMARST and SWRST on the same cycle will cause the 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.
26-22 —	Reserved
21-20 —	Reserved
19 DWT	Double Word Transpose. Setting this bit affects whether the two words within a Dword are transposed when a double-word register is accessed, or when DMA performs a Dword memory transaction or when MOVEDW commands or certain MATH commands are executed. When a double-word CAAM register is read or written by software, the two words are transposed when ! (SSTAR[PLEND] XOR MCFGR[DWT]). That is, • for a Little-Endian platform (PLEND=0), the two words will be transposed if DWT=1. • for a Big-Endian platform (PLEND=1), the two words will be transposed if DWT=0. When CAAM DMA performs a DWord memory access, the two words are transposed when ! (SSTAR[PLEND] XOR MCFGR[DWT] XOR PEO XOR DWSO), where PEO and DWSO are from the appropriate Job Ring Configuration Register (JRCFGR_JR). That is, • for a Little-Endian platform (PLEND=0), the two words will be transposed if DWT=0 and PEO = DWSO. • for a Big-Endian platform (PLEND=1), the two words will be transposed if DWT=1 and PEO = DWSO. For this chip PLEND=0 and DWT=1, so the most-significant half of 64-bit address registers will appear at the lower address (the address given in the register description), unless the endianness has been changed by setting the PEO or DWSO bits. For an explanation of how DWT affects the MOVEDW command, see the MOVE, MOVEB, MOVEDW and MOVE_LEN commands section. For an explanation of how DWT affects the MATH command, see the MATH and MATHI command section.

Table continues on the next page...

Field	Description
18 —	Reserved
17 —	Reserved
16 —	Reserved
15-12 ARCACHE	AXI Read Transaction Attributes. This field provides default values for the generation of the ARCACHE[3:0] interface signals for read transactions. For a general description of ARCACHE signals refer to the AXI3/4 protocol specification. The following functionality, limitations, and extensions exist in this version of CAAM: • ARCACHE[0] (Bufferable): This bit is intended to indicate whether read data may be fetched from an intermediate point in the interconnect or must be fetched from the transaction target (for details see AXI4 specification). • ARCACHE[1] (Cacheable/Modifiable): A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance. • ARCACHE[3:2] (Cache check and allocate controls): This setting and associated signal assertions are irrelevant because this SoC does not have a downstream cache.
11-8 AWCACHE	AXI Write Transaction Attributes. This field provides default values for the generation of the AWCACHE[3:0] interface signals for write transactions. For a general description of AWCACHE signals refer to the AXI3/4 protocol specifications. The following functionality, limitations, and extensions exist for this version of CAAM: • AWCACHE[0] (Bufferable): A setting of 1 indicates the transaction response may be generated from an intermediate point and the transaction may be delayed reaching its final destination (this setting is intended to reduce transaction latency and improve performance). **NOTE:** CAAM always issues non-bufferable writes when it needs to ensure that previously written data using the same AWID has been made visible to other masters after the associated non-bufferable write response is received. For such writes the AWCACHE[0] signal will be forced to 0, independent of the AWCACHE[0] configuration. • AWCACHE[1] (Cacheable/Modifiable): A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance. • AWCACHE[3:2] (Cache check and allocate controls): This setting and associated signal assertions are irrelevant because this SoC does not have a downstream cache.
7-4 AXIPIPE	AXI Pipeline Depth. The AXIPIPE field is a debug field used to adjust the maximum number of outstanding DMA transactions that CAAM is able to queue. Optimal performance will be achieved by retaining the default value of this field. 0000b - Maximum value (since an actual value of zero would effectively disable DMA transactions) 0001b .. 1111b - Upper limit for number of outstanding AXI transactions
3 —	Reserved

Table continues on the next page...

Field	Description
2 LARGE_BURST	Enable Large Bursts. When LARGE_BURST=1, Job Descriptor reads, Shared Descriptor reads and reads of data for the Data FIFO 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. **NOTE:** Changes to LARGE_BURST should be made only when CAAM is not processing jobs.
1 —	Reserved
0 NORMAL_BURST	Normal Burst Size. This field defines the normal burst size for aligned read and write transactions. The normal burst size represents an objective. As needed, CAAM reduces the size by adjusting the actual start and/or end address to meet the functional requirements imposed by the encountered read/write buffer address and size. In addition, CAAM adjusts the end address to enable the use of normal burst aligned addresses in subsequent transactions to maximize performance and lower transaction bandwidth requirements. **NOTE:** NORMAL_BURST was writable in earlier versions of CAAM, but this capability is now obsolete. 0 - Aligned 32 byte burst size target 1 - Aligned 64 byte burst size target

10.13.3 Page 0 SDID Register (PAGE0_SDID)

CAAM enforces control over access to the Secure Memory registers by means of Security Domain Identifier (SDID) values. The owner of a Secure Memory partition is identified by the SDID value stored in the partition's PSDID register. Software can claim an unowned Secure Memory partition by writing into the partition's SMAPR register via a Job Ring register page or via register page 0. At the time the partition is claimed the partition's new owner is recorded by updating the partition's PSDID register.

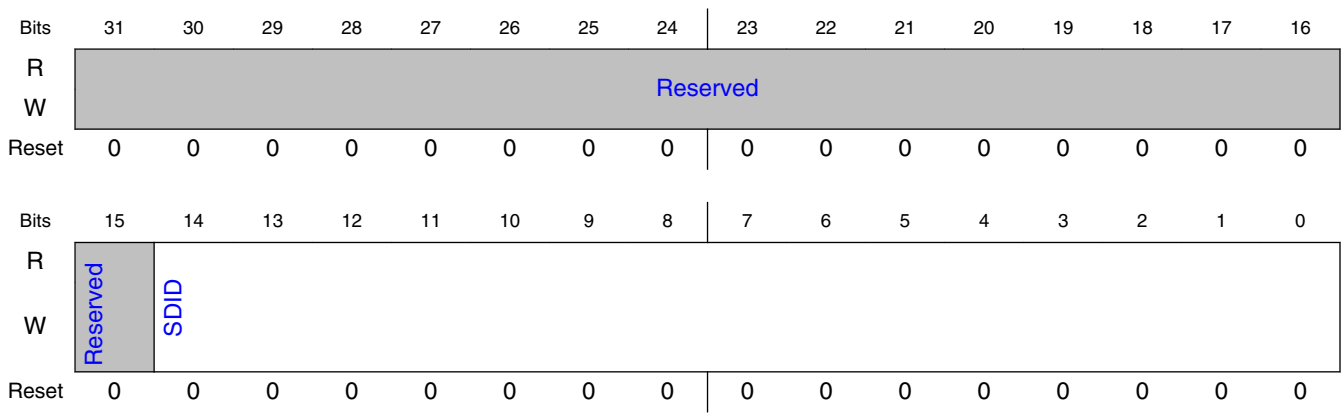
When claiming a partition via a Job Ring register page, the SDID value from the Job Ring's JRaDID register is copied into the partition's PSDID register. When claiming a partition via register page 0, the SDID value from the Page 0 SDID register is copied into the partition's PSDID register. Note that only the CAAM manager (typically the hypervisor and/or TrustZone SecureWorld) can access CAAM register page 0, so only the CAAM manager will be able to claim a Secure Memory partition via a SMAPR address alias in register page 0.

When a Secure Memory partition is claimed by writing into one of the page 0 SMAP register aliases, the SDID value from the PAGE0_SDID register is copied into the partition's PSDID register. Note that bit 4 of the register receives special treatment. See Secure Memory Access Control.

10.13.3.1 Offset

Register	Offset
PAGE0_SDID	8h

10.13.3.2 Diagram



10.13.3.3 Fields

Field	Description
31-15 —	Reserved
14-0 SDID	Security Domain Identifier. When the system software claims a Secure Memory partition by writing (with ns=1) into a partition's Access Permission register via register page 0, hardware indicates the SDID of the nonSecureWorld owner by copying PAGE0_DID[SDID] into the partition's PSDID register but forcing bit 4 to 0 in the partition's PSDID register. (This ensures that the partition is owned by TrustZone non-SecureWorld.) When Trust_Zone Secure_World claims a secure memory partition by writing (with ns=0) into a partition's Access Permission register via register page 0, hardware indicates the partition owner by copying PAGE0_DID[SDID] into the partition's PSDID register. (If bit 4 is 1, the partition will be owned by TrustZone SecureWorld. But SecureWorld can claim a partition for nonSecureWorld by setting bit 4 to 0.) Note that the value 7FFFh should not be written to this bit field.

10.13.4 Security Configuration Register (SCFGR)

The Security Configuration Register is primarily used to set security-related mode bits enabling the controlled transition from special boot-time to normal run-time operating modes. At POR, the SCFGR is reset to 0.

When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in deterministic mode. This allows the secure boot software to run deterministic tests on the RNG and its State Handle 0 logic. Once the tests have been completed, the secure 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 always used.

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 secure 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 secure boot blob. On subsequent boot cycles the protected reference metrics would be obtained by decapsulating the private secure boot 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.

To prevent later stages of software from decrypting blobs used by boot software or key provisioning software, and to prevent later stages of software from generating counterfeit blobs that would be accepted as genuine by boot software or provisioning software on later boot cycles, the boot software or provisioning software or the initial boot image should set one or both of the PRIBLOB bits before passing control to later software. As illustrated in Figure 10-23, typically the secure boot 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 secure boot blobs could no longer be encapsulated or decapsulated. The secure boot software 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 secure boot software blobs or either type of private provisioning blobs.

When deploying an encrypted boot environment the cryptographic separation of private blob types should be applied. The Data Encryption Key (DEK) must be initially encapsulated as a private provisioning type blob (PRIBLOB=01 or PRIBLOB=10) and the run time software should only be able to create normal operation blobs (PRIBLOB=11). This separation of private blob types avoids any undetected modification or replacement of Data Encryption Key (DEK) blobs.

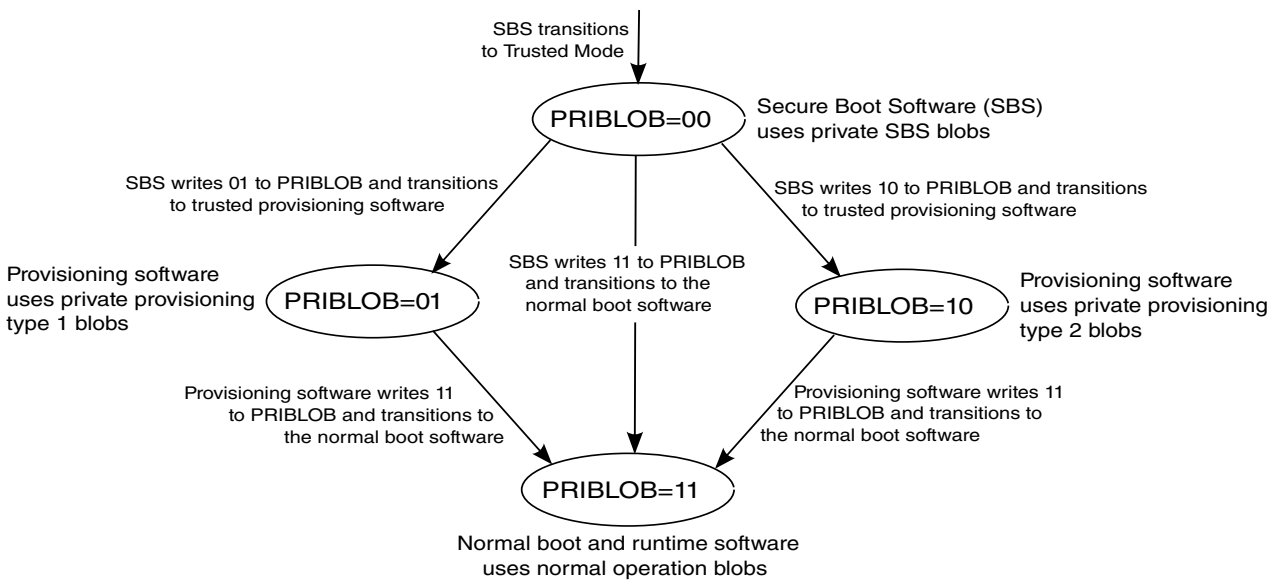
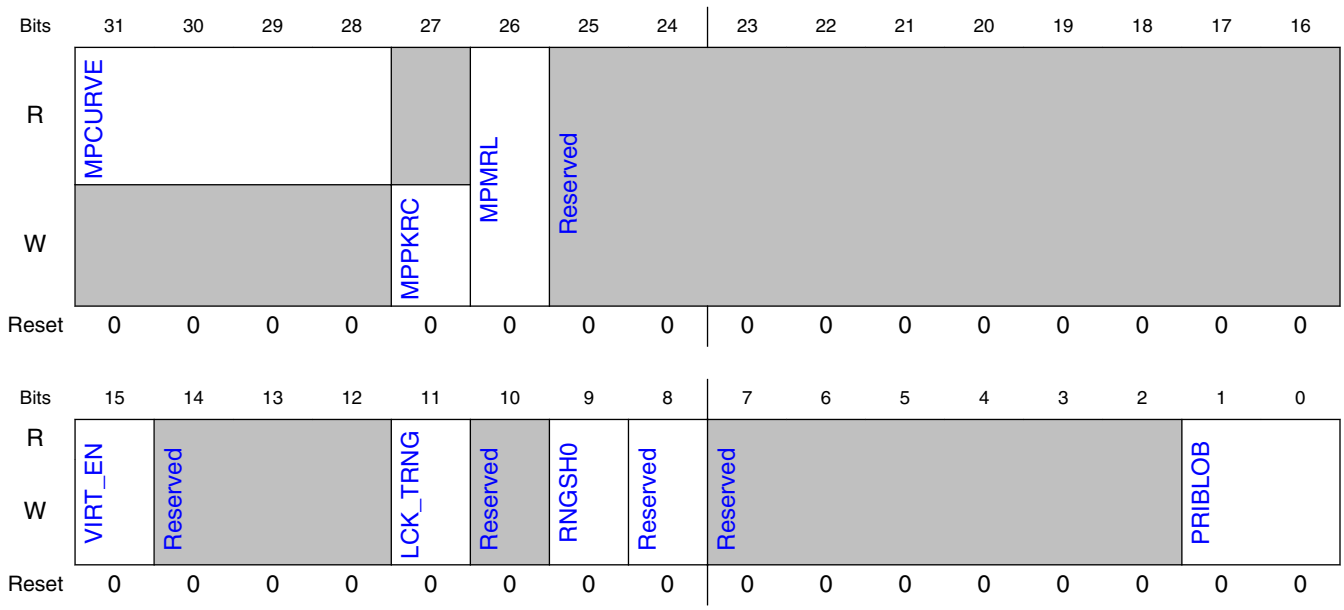


Figure 10-23. Process for Managing Private Blobs

10.13.4.1 Offset

Register	Offset
SCFGR	Ch

10.13.4.2 Diagram



10.13.4.3 Fields

Field	Description
31-28 MPCURVE	Manufacturing Protection Curve. This shows the elliptic curve that was selected when the MPPrivK Generation protocol was run.
27 MPPKRC	Manufacturing Protection Private Key Register Clear. Writing a 1 to this bit clears the Manufacturing Protection Private Key Register.
26 MPMRL	Manufacturing Protection Message Register Lock. Writing a 1 to this bit locks the Manufacturing Protection Message Register for writing. The register remains locked until the next POR.
25-16 —	Reserved
15 VIRT_EN	Virtualization enable. Virtualization is disabled by default. Writing a 1 to this bit enables Job Ring virtualization. When Job Ring virtualization in enabled, the Start_JRa bits in the JRSTART register must be used to switch between writing the Job Ring registers in register page 0 and writing the Job Ring registers in register pages 1....3. **NOTE:** The LAMTD bit in the JRaDID register cannot be written unless VIRT_EN is 1. 0 - Disable job ring virtualization 1 - Enable job ring virtualization
14-12 —	Reserved

Table continues on the next page...

Field	Description
11 LCK_TRNG	Lock TRNG Program Mode. Writing a 1 to this bit locks the TRNG. That is, when this bit is set TRNG can't go into program mode. If it is in program mode when this bit is set, the TRNG will immediately leave program mode. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.
10 —	Reserved
9 RNGSH0	Random Number Generator State Handle 0. 0 - When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in any mode. RNGSH0 is set to 0 only for testing. 1 - When RNGSH0 is 1, RNG DRNG State Handle 0 cannot be instantiated in deterministic (test) mode. RNGSHO should be set to 1 before the RNG is instantiated. 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 —	Reserved
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 secure boot software blobs 01 - Private provisioning type 1 blobs 10 - Private provisioning type 2 blobs 11 - Normal operation blobs

10.13.5 Job Ring a DID Register - most significant half (JR0DID_MS - JR2DID_MS)

There is one JRaDID_MS register per Job Ring. This Register is used to indicate the Security Domain (SDID) that currently owns the Job Ring and to specify the TrustZone SecureWorld, DID and ICID values that the CAAM DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular Job Ring.

TrustZone SecureWorld can reserve a Job Ring for itself by setting the TZ_OWN bit to 1. Note that TZ_OWN can be set to 1 only if the register is written using a SecureWorld bus transaction. If TZ_OWN=1, the JRaDID_MS and JRaDID_LS registers can be written only via a SecureWorld bus transaction. PRIM_TZ can be set to 1 only if TZ_OWN=1 and the register is written using a SecureWorld bus transaction. That is, only a

SecureWorld-owned Job Ring can cause CAAM to assert a SecureWorld bus transaction, but a SecureWorld-owned Job Ring can be set to assert either SecureWorld or Non-SecureWorld transactions.

The Job Ring register pages (pages 1 .. 3) can be accessed only via bus transactions using TrustZone and DID values that match the values in the PRIM_TZ and PRIM_DID fields in that Job Ring's JRaDID_MS register. Bus transactions that use other TrustZone or DID values will be ignored.

JRaDID_MS contains a USE_OUT field that enables a second set of ICID and DID values. When USE_OUT=1, this Job Ring's **data** write transactions will assert TrustZone Non-SecureWorld, along with the OUT_DID and OUT_ICID values from JRSDID_LS. All other bus transactions, including all reads, descriptor write-backs and job completion status writes will assert the PRIM_ICID, PRIM_ICID and **not** PRIM_TZ values from JRaDID_MS. When USE_OUT=0, all bus transactions performed on behalf of this Job Ring will use the PRIM_ICID, PRIM_ICID and **not** PRIM_TZ values from JRSDID_MS.

This register also contains a bit (AMTD) that grants permission for Trusted Descriptors to be created in this Job Ring. Note that there is a lock bit (LAMTD) specifically for the AMTD field. Once LAMTD is set to 1 the LAMTD and AMTD bits cannot be modified until the next POR. If the Job Ring is owned by TrustZone SecureWorld (TZ_OWN=1), any Trusted Descriptor created in this Job Ring is marked as a TrustZone Trusted Descriptor (see TDES field in HEADER command. If the Job Ring is not owned by TrustZone SecureWorld, any Trusted Descriptor created in the job ring is marked as a nonTrustZone Trusted Descriptor.

The Security Domain Identifer (SDID) bits are mixed into
- the JDKEK or TDKEK to encrypt SDID-specific black keys
- the TDSK to sign SDID-specific Trusted Descriptors
- the BKEK to encrypt/decrypt SDID-specific blobs

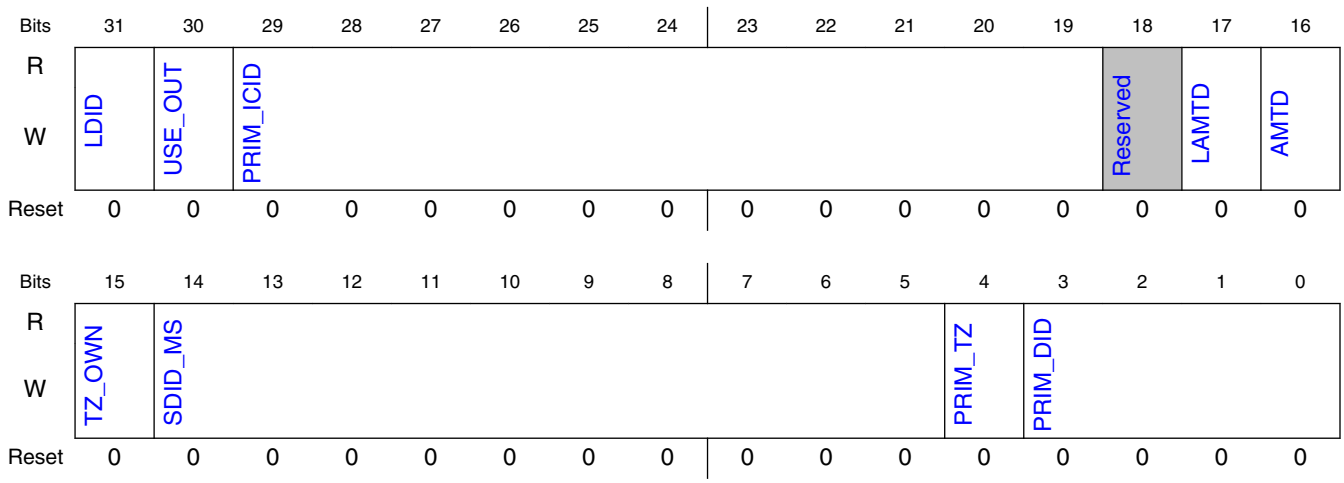
When USE_OUT=0 and PRIM_TZ=0, the SDID consists of the concatenation of the SDID_MS, PRIM_TZ and PRIM_DID fields. When USE_OUT=0 and PRIM_TZ=1, all bits of the SDID_MS field are forced to zeros, so SDID consists of the concatenation of 0000000000b, and the PRIM_TZ and PRIM_DID fields. Therefore, the only SecureWorld Black Keys, Blobs, and Trusted Descriptors that are accessible within this DID-specific SecureWorld are those corresponding to this single SDID value. When USE_OUT=1, the SDID consists of the concatenation of the SDID_MS field, 0b (non-SecureWorld) and the OUT_DID field.

The PRIM_TZ, PRIM_DID and OUT_DID fields are typically written at boot time and then locked by setting LDID to 1. Once LDID is set to 1, LDID, PRIM_TZ, PRIM_DID and OUT_DID cannot be modified until the next POR.

10.13.5.1 Offset

Register	Offset	Description
JR0DID_MS	10h	used with Job Ring 0
JR1DID_MS	18h	used with Job Ring 1
JR2DID_MS	20h	used with Job Ring 2

10.13.5.2 Diagram



10.13.5.3 Fields

Field	Description
31 LDID	Lock DIDs. Once LDID has been set to 1, no further changes can be made to the LDID, PRIM_DID, PRIM_TZ, or OUT_DID fields until reset. The SDID_MS field is not locked, and the AMTD is locked only if LAMTD=1.
30 USE_OUT	USE_OUT. When USE_OUT=0, all CAAM DMA transactions performed on behalf of this Job Ring assert the PRIM_ICID, PRIM_TZ and PRIM_DID values. When USE_OUT=1, data output DMA transactions (i.e. writing data to memory) performed on behalf of this Job Ring assert the OUT_ICID and OUT_DID values, and assert ns as 1 (i.e. TrustZone non-SecureWorld). All other CAAM DMA transactions performed on behalf of this Job Ring (including job completion status writes and writes back to job descriptors) assert the PRIM_ICID, PRIM_TZ and PRIM_DID values.
29-19 PRIM_ICID	Primary ICID. This ICID value is asserted during certain CAAM DMA transactions. (See JRaDID_MS[USE_OUT].)
18 —	Reserved

Table continues on the next page...

Field	Description
17 LAMTD	Lock AMTD. Once LAMTD has been set, no further changes can be made to the AMTD field. Note that the LAMTD bit cannot be written unless virtualization mode is enabled (SCFGR[VIRT_EN]=1).
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 options specified in the SIGNATURE command at the end of the descriptor determine whether DECO will execute the commands in the descriptor, or append a signature to the descriptor, or both. If the Job Ring is owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone Trusted Descriptor, otherwise the descriptor will be treated as a non-SecureWorld Trusted 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.
15 TZ_OWN	TrustZone SecureWorld. This bit can be written only by TrustZone SecureWorld (i.e. a bus transaction with ns=0). If TZ_OWN=1, this Job Ring is owned by TrustZone SecureWorld. If TZ_OWN=0 this Job Ring is owned by non-SecureWorld.
14-5 SDID_MS	Security Domain Identifier most significant bits. When USE_OUT=0 and PRIM_TZ=0, these bits are prepended to the PRIM_TZ and PRIM_DID fields to form a 15-bit Security Domain Identifier (SDID). Note that there is only one TrustZone SecureWord security domain per DID, so when USE_OUT=0 and PRIM_TZ=1 the SDID_MS is forced to all 0s so there is only one SDID value per DID. When USE_OUT=1 the value in the SDID_MS field is concatenated with a 0 bit and the OUT_DID field to form the SDID. The 15-bit SDID value is used to tag Black Keys, Blobs, and Trusted Descriptors so they can be used only by this security domain. When claiming a Secure Memory partition, the SDID value is copied into the partition's PSDID register to indicate that the partition is owned by this Security Domain.
4 PRIM_TZ	Primary TZ. This field specifies the TrustZone SecureWorld/non-SecureWorld value asserted during certain CAAM DMA transactions (see field JRaDID_MS[USE_OUT]). This field also specifies whether the registers that are specific to a particular Job Ring can be read or written only by a bus master operating in TrustZone SecureWorld. These registers include the Job Ring configuration registers, the interrupt registers, the CAAM Secure Memory Access Permissions and Secure Memory Access Group registers and the ring buffer registers. Note that PRIM_TZ = 1 means TrustZone SecureWorld and PRIM_TZ = 0 means TrustZone **non**-SecureWorld. PRIM_TZ can be written to 1 only via a TrustZone SecureWorld transaction and only if TZ_OWN = 1. Consequently, only TrustZone SecureWorld can cause a CAAM Job Ring DMA transaction to assert TrustZone SecureWorld.
3-0 PRIM_DID	Job Ring Owner's DID. This field defines the DID 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 Group registers and the ring buffer registers. This field also specifies the DID value asserted during certain CAAM DMA transactions (see field JRaDID_MS[USE_OUT]).

10.13.6 Job Ring a DID Register - least significant half (JR0DID_LS - JR2DID_LS)

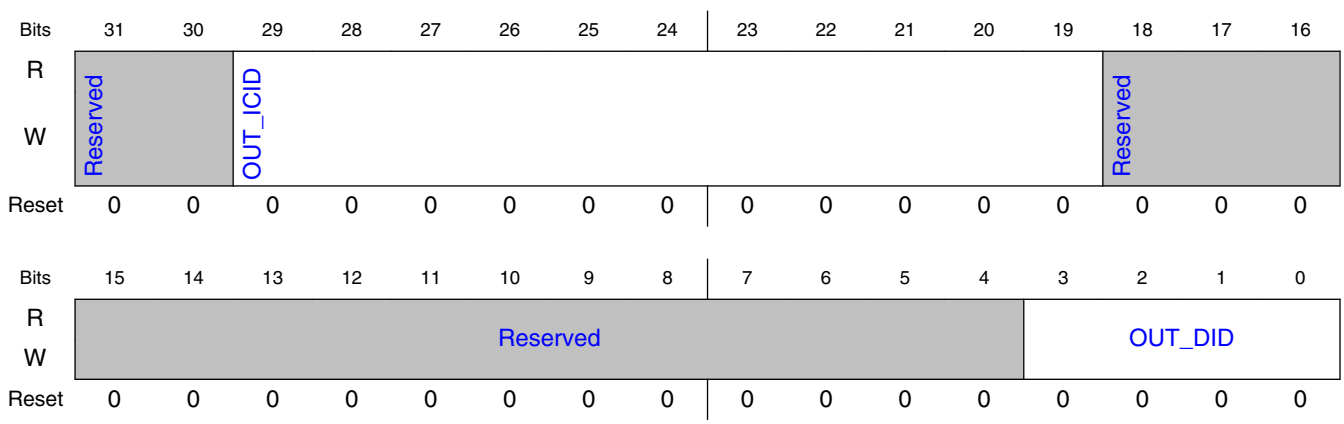
There is one JRaDID_LS register per Job Ring. This register contains the OUT_ICID and OUT_DID values that are enabled when JRaDID_MS[USE_OUT]=1. When USE_OUT=1 this Job Ring's data write transactions will assert the OUT_DID and OUT_ICID values from JRaDID_LS but ignore JRaDID_MS[PRIM_TZ] and instead always assert TrustZone NonSecure World (ns=1). All other bus transactions, including all read transactions and descriptor write-back and job completion status writes will assert

the PRIM_ICID value and ns = **not** PRIM_TZ value from JRaDID_MS. When USE_OUT=0 all bus transactions performed on behalf of this Job Ring, including input data reads, will use the PRIM_ICID and ns = **not** PRIM_TZ value from JRaDID_MS. If JRaDID_MS[TZ_OWN]=1, the JRaDID_MS and JRaDID_LS registers can be written only via a SecureWorld bus transaction.

10.13.6.1 Offset

Register	Offset	Description
JR0DID_LS	14h	used with Job Ring 0
JR1DID_LS	1Ch	used with Job Ring 1
JR2DID_LS	24h	used with Job Ring 2

10.13.6.2 Diagram



10.13.6.3 Fields

Field	Description
31-30 —	Reserved
29-19 OUT_ICID	Job Ring Output ICID. This field defines the ICID value that, if JRaDID_MS[USE_OUT]=1, will be asserted for data write DMA transactions performed on behalf of this Job Ring. All read transactions, including input data, keys, scatter/gather tables and descriptors and job completion status write transactions will assert JRaDID_MS[PRIM_ICID]. If USE_OUT=0, all bus transactions performed on behalf of this Job Ring will assert PRIM_ICID.
18-4	Reserved

Table continues on the next page...

Field	Description
—	
3-0 OUT_DID	Output DID. This field defines the DID value that, if JRaDID_MS[USE_OUT]=1, will be asserted for data write DMA transactions performed on behalf of this Job Ring. Descriptor write-back and job completion status write transactions will assert JRaDID_MS[PRIM_DID]. All read transactions, including input data, keys, scatter/gather tables and descriptors and job completion status write transactions will assert JRaDID_MS[PRIM_DID]. If USE_OUT=0, all bus transactions performed on behalf of this Job Ring will assert PRIM_DID. Once the LDID bit in the JRaDID_MS register has been set to 1, until reset no further changes can be made to the OUT_DID field.

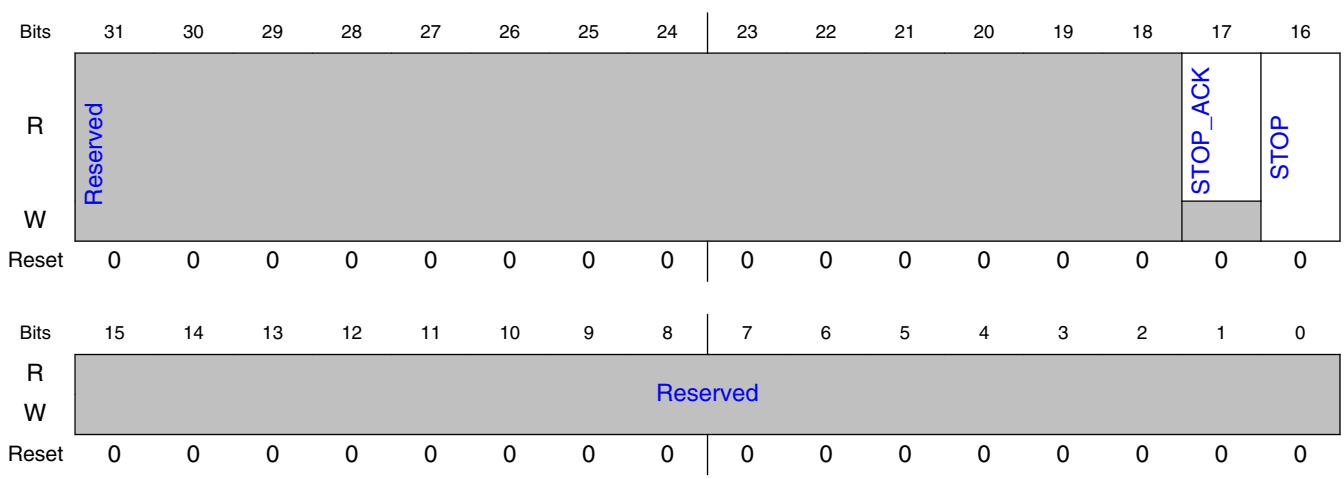
10.13.7 Debug Control Register (DEBUGCTL)

The DEBUGCTL Register is used to stop CAAM from processing jobs so that a consistent read of the debug registers can be performed.

10.13.7.1 Offset

Register	Offset
DEBUGCTL	58h

10.13.7.2 Diagram



10.13.7.3 Fields

Field	Description
31-18 —	Reserved
17 STOP_ACK	STOP_ACK will assert when the job queue controller acknowledges that it is stopped.
16 STOP	STOP is written to 1 to request that CAAM stop processing jobs. This is intended to be a graceful halt. CAAM will shut down in such a way that it will be able to resume processing where it left off once software has finished reading the debug registers. Note that the RTIC watchdog timer will continue to run during the halt. It is recommended that the DECO watchdog timer be turned off (see Master Config Register) prior to stopping CAAM in order to prevent a possible watchdog error from a job in a stopped DECO. When STOP is asserted, the DECO Availability register can be used to monitor which DECOs are stopped or are available.
15-0 —	Reserved

10.13.8 Job Ring Start Register (JRSTARTR)

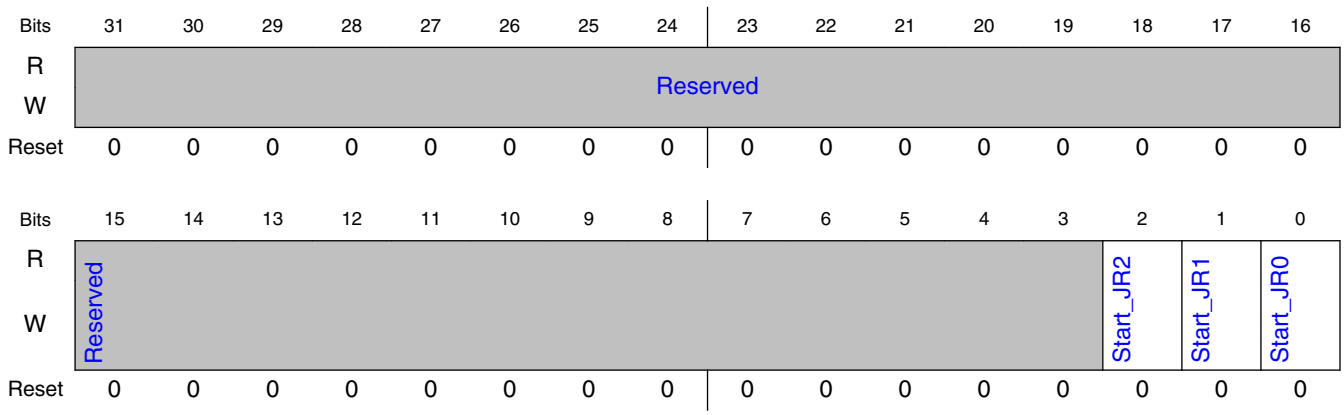
The Job Ring Start register is used by the system software or TrustZone SecureWorld when configuring a Job Ring for a new user. Before the Job Ring is configured for a new user, the Job Ring must be in stop mode. Before the new user can set up the Job Ring and begin using it, the Job Ring must be in start mode.

The Job Ring Start register is not used and is not writable when virtualization mode is disabled in the Security Configuration register. All bits in this register will remain at the default 0 value when virtualization mode is disabled.

10.13.8.1 Offset

Register	Offset
JRSTARTR	5Ch

10.13.8.2 Diagram



10.13.8.3 Fields

Field	Description
31-3 —	Reserved
2 Start_JR2	Start Job Ring 2. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 2 is allocated to TrustZone SecureWorld (JR2DID[TZ]=1), Start_JR2 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0. 0 - Stop Mode. The JR2DID register and the SMVBA register for Job Ring 2 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 are NOT accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2DID[TZ]=1), the JR2DID and SMVBA register can be written only via a bus transaction that has ns=0. 1 - Start Mode. The JR2DID register and the SMVBA register for Job Ring 2 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 ARE accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2DID[TZ]=1), then the SMVBA, IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 2 can be written only via a bus transaction that has ns=0.
1 Start_JR1	Start Job Ring 1. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 1 is allocated to TrustZone SecureWorld (JR1DID[TZ]=1), Start_JR1 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0. 0 - Stop Mode. The JR1DID register and the SMVBA register for Job Ring 1 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 are NOT accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1DID[TZ]=1), the JR1DID and SMVBA register can be written only via a bus transaction that has ns=0. 1 - Start Mode. The JR1DID register and the SMVBA register for Job Ring 1 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 ARE accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1DID[TZ]=1), then the SMVBA, IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 1 can be written only via a bus transaction that has ns=0.
0 Start_JR0	Start Job Ring 0. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 0 is allocated to TrustZone SecureWorld (JR0DID[TZ]=1), Start_JR0 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.

Field	Description
	0 - Stop Mode. The JR0DID register and the SMVBA register for Job Ring 0 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 are NOT accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0DID[TZ]=1), the JR0DID and SMVBA register can be written only via a bus transaction that has ns=0. 1 - Start Mode. The JR0DID register and the SMVBA register for Job Ring 0 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 ARE accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0DID[TZ]=1), then the SMVBA, IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 0 can be written only via a bus transaction that has ns=0.

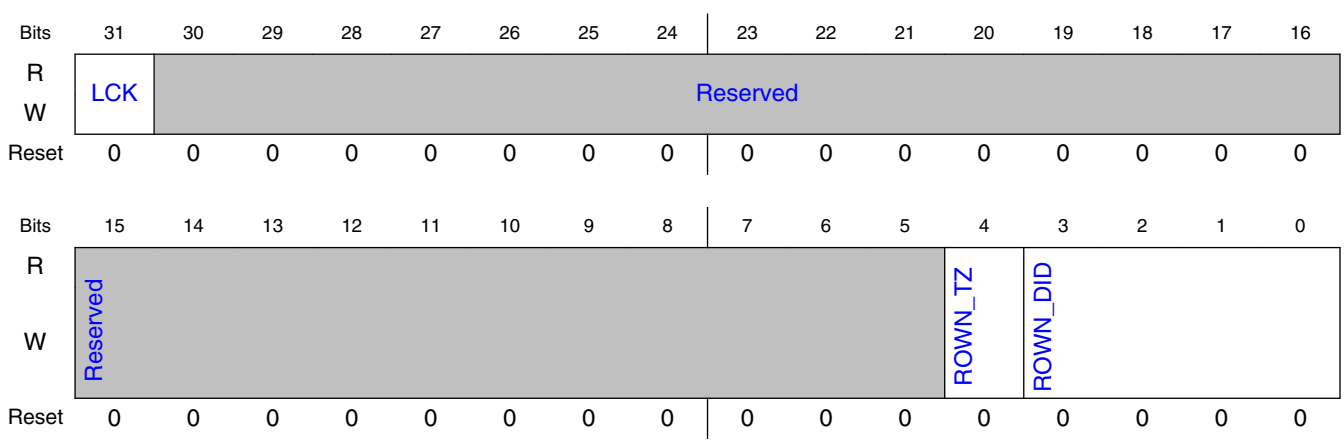
10.13.9 RTIC OWN Register (RTIC_OWN)

This register is used to specify the IP Register Bus DID and TrustZone World that must be asserted by the processor in order to read or write the Real Time Integrity Checker (RTIC) registers (excluding the RTIC DID registers, which are accessible only to the Manager Processor). This register is typically written at boot time and then locked.

10.13.9.1 Offset

Register	Offset	Description
RTIC_OWN	60h	controls access to the RTIC register page

10.13.9.2 Diagram



10.13.9.3 Fields

Field	Description
31 LCK	RTIC OWN Lock. Once LCK has been set, no further changes can be made to the RTIC_OWN register (including changes to LCK) or the four RTIC DID registers until the next POR.
30-5 —	Reserved
4 ROWN_TZ	RTIC Owner's TZ. This field defines the TrustZone World (SecureWorld = 1) of the bus master that is permitted to read or write the register in the RTIC page. The ROWN_TZ bit cannot be written with a non-SecureWorld bus transaction. When the RTIC_OWN register is written with a SecureWorld bus transaction the ROWN_TZ bit can be written to either 0 or 1.
3-0 ROWN_DID	RTIC Owner's DID RTIC Owner's DID. This field defines the DID of the bus master that is permitted to read or write the registers in the RTIC page.

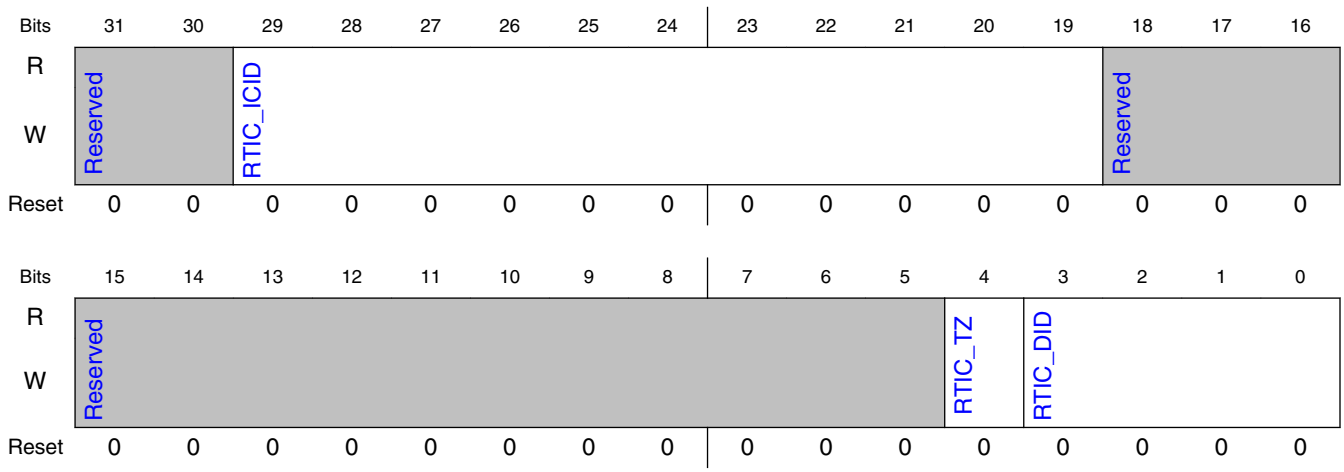
10.13.10 RTIC DID Register for Block a (RTICA_DID - RTICD_DID)

There is one RTIC_DID register per RTIC hash block. The RTIC DID Register is used to specify the AXI bus DID and ICID values that the CAAM DMA asserts when reading a particular RTIC hash block from memory external to CAAM.

10.13.10.1 Offset

Register	Offset	Description
RTICA_DID	64h	used with Block A
RTICB_DID	6Ch	used with Block B
RTICC_DID	74h	used with Block C
RTICD_DID	7Ch	used with Block D

10.13.10.2 Diagram



10.13.10.3 Fields

Field	Description
31-30 —	Reserved
29-19 RTIC_ICID	RTIC ICID. This field defines the ICID value asserted when RTIC accesses the memory addresses in hash region a.
18-5 —	Reserved
4 RTIC_TZ	RTIC_TZ. This field defines the TrustZone TZ value (SecureWorld = 1) asserted when RTIC accesses memory. The RTIC_TZ bit cannot be set to 1 using a non-SecureWorld bus transaction or if RTIC is under control of non-SecureWorld (RTIC_OWN[ROWN_TZ]=0). When RTIC_OWN[ROWN_TZ]=1 and the RTIC_DID register is written with a SecureWorld bus transaction the RTIC_TZ bit can be written to either 0 or 1.
3-0 RTIC_DID	RTIC DID. This field defines the DID value asserted when RTIC accesses the memory addresses in hash region a.

10.13.11 DECO Request Source Register (DECORSR)

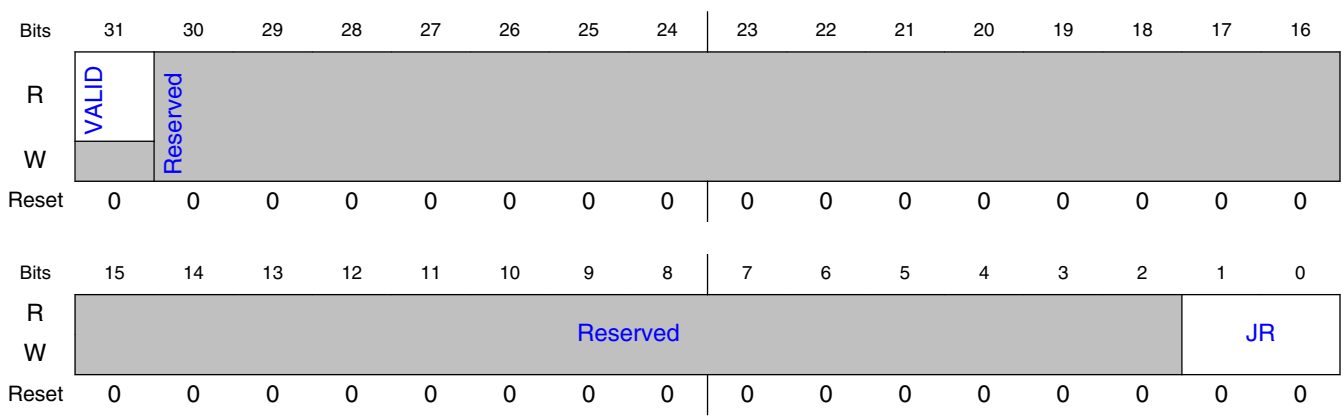
The DECO Request Source Register is used to indicate a particular Job Ring whose JRaDID register will be used to supply the DID, ICID, TZ_OWN and SDID values and whose JRSMBA register will be used to supply the Secure Memory base address when descriptor commands are executed under direct software control. The selected Job Ring

will be used for all DECOs that are presently under direct software control regardless of whether those DECOs were requested via a single write or via multiple writes to the DECO Request Register. The DECO Request Source Register is writable only when all bits in the DECO Request Register are 0 (i.e. no DECOs are requested or already under direct software control). If the Job Ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

10.13.11.1 Offset

Register	Offset
DECORSR	94h

10.13.11.2 Diagram



10.13.11.3 Fields

Field	Description
31 VALID	Valid. This bit will be set to 1 to indicate that the JR field contains a valid Job Ring number. If an invalid Job Ring number is written to JR, the VALID bit will be 0. The VALID bit will always remain 0 if virtualization mode is disabled.
30-2 —	Reserved
1-0 JR	Job Ring number. This Job Ring's JRaDID register will be used to supply the DID, ICID, TZ_OWN and SDID values and this Job Ring's JRSMBA register will be used to supply the Secure Memory base address when descriptor commands are executed under direct software control. If the specified Job Ring

Field	Description
	is not implemented or if virtualization is enabled and the specified Job Ring is not in start mode, the JR field will not be changed and the VALID bit will remain 0. Note that it is not possible to select a job ring that has PRIM_TZ=1. Regardless of whether ns=1 or 0, if the write to DECORSR specifies such a Job Ring the VALID bit will remain 0.

10.13.12 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. This interface would normally be used only for debugging and testing purposes since it is not as efficient as the Job Ring Interface. The procedure for directly accessing the DECO/CCB block is described in detail in Register-based service interface.

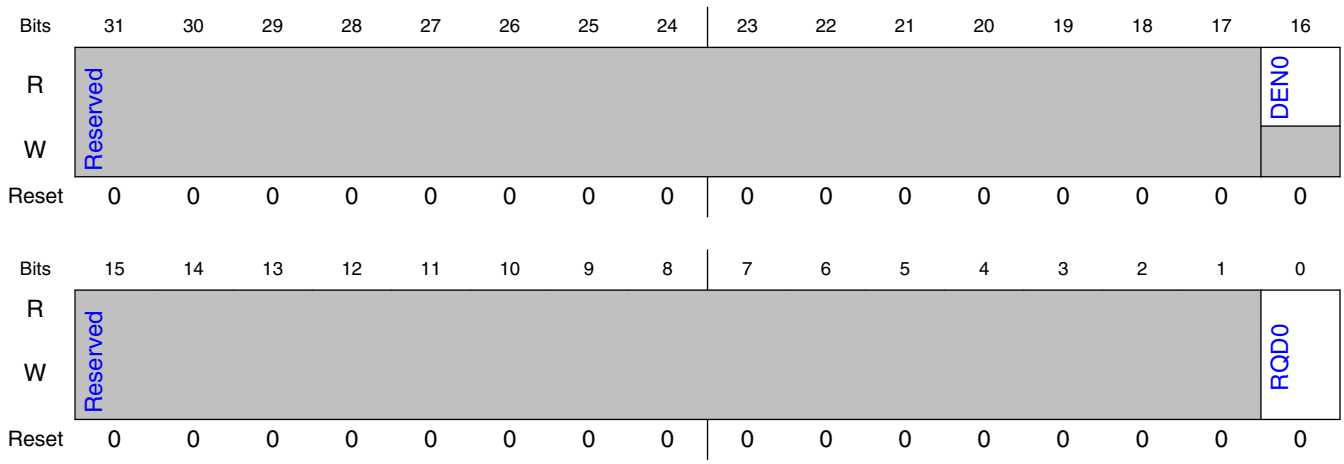
Note that, unless virtualization mode is disabled in the Security Configuration register, a Job Ring that is in the start mode must be selected via the DECO Request Source Register prior to requesting a DECO via DECORR. If the DECO Request Source Register's VALID bit is not set, or if the selected Job Ring is not in start mode, the DECO Request Register cannot be written. The DID, TZ and SDID value from the selected job ring's JRaDID register and the Secure Memory base address from the Job Ring's JRSMBA register will be used when descriptor commands are executed under direct software control. If the Job Ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

If virtualization mode is disabled the DECO DID registers supply the DID values used when descriptor commands are executed under direct software control, and an all-zero SDID is used. The DECO Request Source register is not used when virtualization is disabled.

10.13.12.1 Offset

Register	Offset
DECORR	9Ch

10.13.12.2 Diagram



10.13.12.3 Fields

Field	Description
31-17 —	Reserved
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
0 RQD0	This bit is set by software to request direct access to DECO 0/CCB 0. It cannot be cleared until the direct access operation is complete or CAAM gets a software reset.

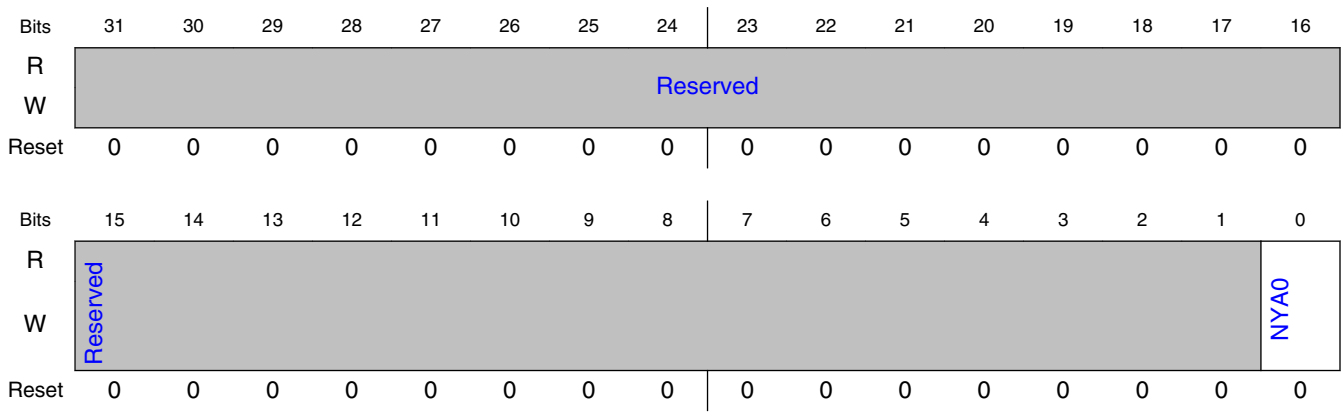
10.13.13 DECO Availability Register (DAR)

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. If STOP is asserted in the DEBUG Control register, the DECO Availability Register cannot be written. While STOP is asserted DECO Availability provides a status for whether each DECO is stopped or available. Any bit that is zero in this case indicates a DECO that is still running and needs to stop before the Debug Control Register can assert STOP_ACK.

10.13.13.1 Offset

Register	Offset
DAR	120h

10.13.13.2 Diagram



10.13.13.3 Fields

Field	Description
31-1 —	Reserved
0 NYA0	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.

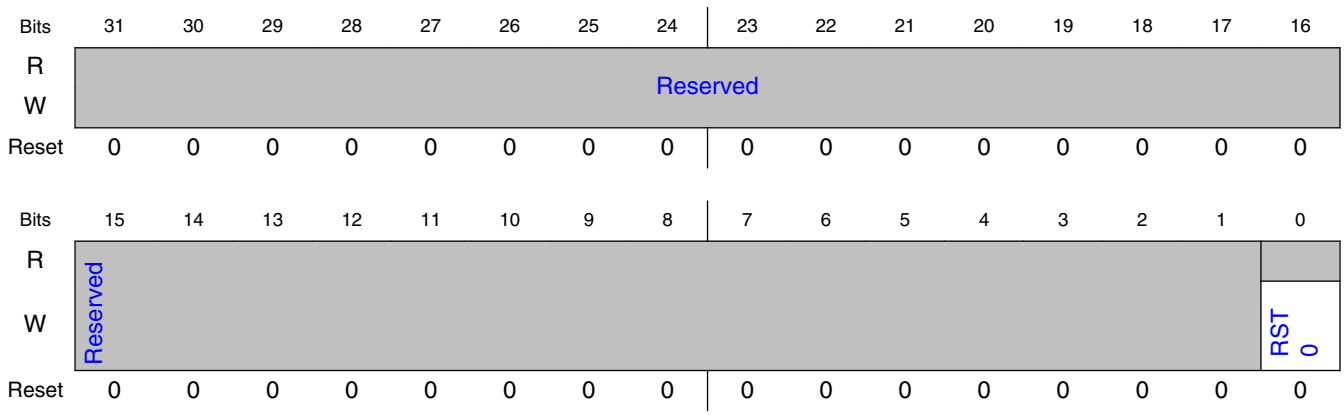
10.13.14 DECO Reset Register (DRR)

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.

10.13.14.1 Offset

Register	Offset
DRR	124h

10.13.14.2 Diagram



10.13.14.3 Fields

Field	Description
31-1 —	Reserved
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.

10.13.15 Job Ring a Secure Memory Virtual Base Address Register (JR0SMVBAR - JR2SMVBAR)

The Job Ring Secure Memory Virtual Base Address Register is used by the system software to specify the base address of Secure Memory within the virtual address space used by a particular Job Ring. There is one copy of the register per Job Ring. All pages of Secure Memory are expected to be mapped contiguously starting at the specified virtual

base address. CAAM's DMA uses this base address, together with the size of Secure Memory, to determine whether to steer accesses to Secure Memory or to external memory.

10.13.15.1 Offset

Register	Offset	Description
JR0SMVBAR	184h	used with Job Ring 0
JR1SMVBAR	18Ch	used with Job Ring 1
JR2SMVBAR	194h	used with Job Ring 2

10.13.15.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SMVBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SMVBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.15.3 Fields

Field	Description
31-0 SMVBA	Secure Memory Virtual Base Address. This field contains the upper bits of the base address of Secure Memory in this Job Ring's virtual address space. Since the base address of Secure Memory must be on a 64 kbyte boundary, the least significant 16 bits are omitted. That is, the full address is SMVBA followed by 0000h.

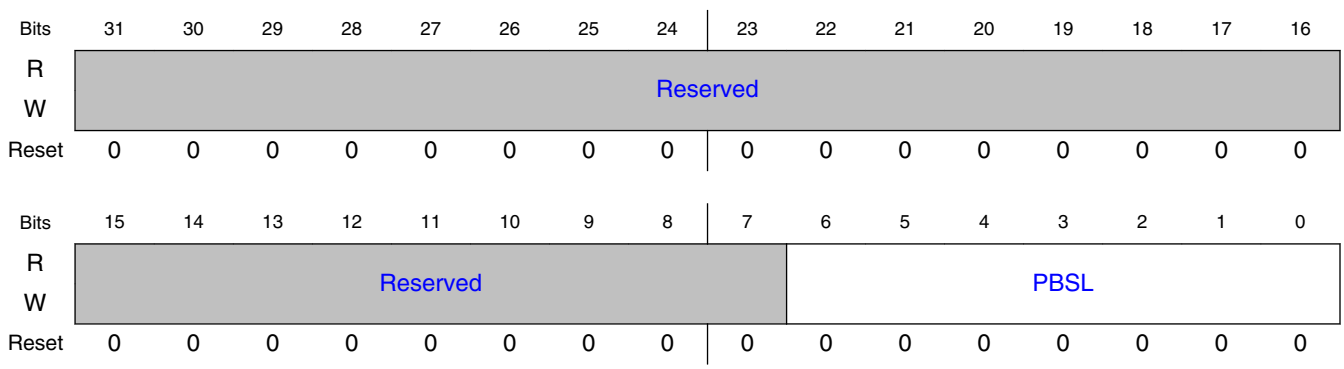
10.13.16 Peak Bandwidth Smoothing Limit Register (PBSL)

The Peak Bandwidth Smoothing Limit Register is used to limit the maximum bus bandwidth consumed by CAAM.

10.13.16.1 Offset

Register	Offset
PBSL	220h

10.13.16.2 Diagram



10.13.16.3 Fields

Field	Description
31-7 —	Reserved
6-0 PBSL	Whenever the number of outstanding AXI read bursts exceeds the value programmed in this field, the Job Rings will be prevented from issuing additional AXI reads. While the number of outstanding AXI read burst exceeds the PBSL, DECOs may continue to issue additional AXI read requests. The Job Rings will be allowed to issue additional AXI reads only when the number of outstanding AXI read bursts drops to, or below, the PBSL. Throttling the AXI reads reduces the CAAM peak bandwidth on the AXI bus, and giving priority to DECOs improves CAAM performance when CAAM is heavily loaded with jobs. A limit of PBSL=0 indicates that no AXI read smoothing will be performed.

10.13.17 DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS)

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to CAAM internal blocks. These assignments are made via hardwired signals and are SoC-specific. The value of each 8-bit field indicates the internal ID of the CAAM block that will use the AXI ID corresponding to the field. For example, AID2BID=00001000 means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMAn AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMA. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

The CAAM internal block IDs are encoded as follows:

Internal Block ID	Internal Logic Block
00000001b	Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.)
00001000b	DECO0
All other values are reserved.	

NOTE

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

10.13.17.1 Offset

Register	Offset	Description
DMA0_AIDL_MAP_MS	240h	Mapping for DMA AXI IDs 7 ... 4

10.13.17.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID7_BID								AID6_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID5_BID								AID4_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.17.3 Fields

Field	Description
31-24 AID7_BID	This field shows the CAAM Block ID that uses AXI ID 7.
23-16 AID6_BID	This field shows the CAAM Block ID that uses AXI ID 6.
15-8 AID5_BID	This field shows the CAAM Block ID that uses AXI ID 5.
7-0 AID4_BID	This field shows the CAAM Block ID that uses AXI ID 4.

10.13.18 DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS)

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.18.1 Offset

Register	Offset	Description
DMA0_AIDL_MAP_LS	244h	Mapping for DMA AXI IDs 3 ... 0

10.13.18.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID3_BID								AID2_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID1_BID								AID0_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.18.3 Fields

Field	Description
31-24 AID3_BID	This field shows the CAAM Block ID that uses AXI ID 3.
23-16 AID2_BID	This field shows the CAAM Block ID that uses AXI ID 2.
15-8 AID1_BID	This field shows the CAAM Block ID that uses AXI ID 1.
7-0 AID0_BID	This field shows the CAAM Block ID that uses AXI ID 0.

10.13.19 DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS)

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

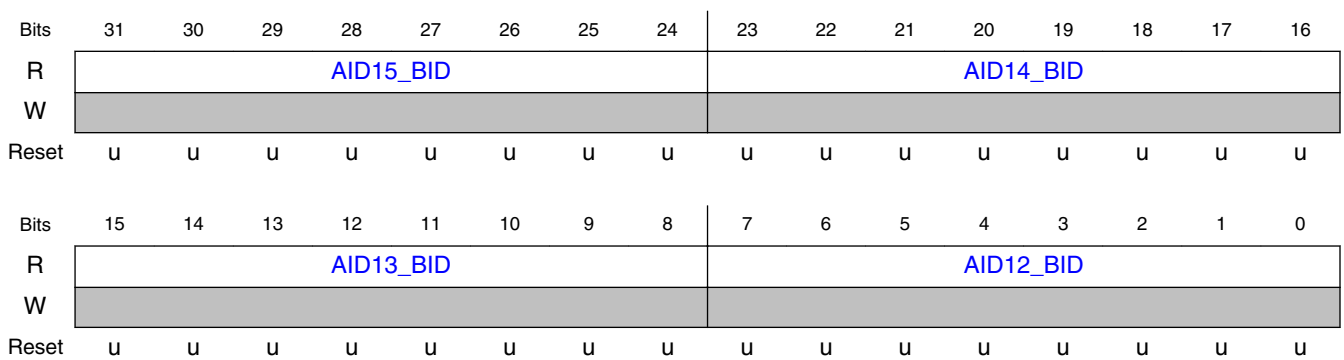
NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.19.1 Offset

Register	Offset	Description
DMA0_AIDM_MAP_MS	248h	Mapping for DMA AXI IDs 15 ... 12

10.13.19.2 Diagram



10.13.19.3 Fields

Field	Description
31-24 AID15_BID	This field shows the CAAM Block ID that uses AXI ID 15.
23-16 AID14_BID	This field shows the CAAM Block ID that uses AXI ID 14.
15-8 AID13_BID	This field shows the CAAM Block ID that uses AXI ID 13.
7-0 AID12_BID	This field shows the CAAM Block ID that uses AXI ID 12.

10.13.20 DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS)

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.20.1 Offset

Register	Offset	Description
DMA0_AIDM_MAP_LS	24Ch	Mapping for DMA AXI IDs 11 ... 8

10.13.20.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID11_BID								AID10_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID9_BID								AID8_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.20.3 Fields

Field	Description
31-24 AID11_BID	This field shows the CAAM Block ID that uses AXI ID 11.

Table continues on the next page...

Field	Description
23-16 AID10_BID	This field shows the CAAM Block ID that uses AXI ID 10.
15-8 AID9_BID	This field shows the CAAM Block ID that uses AXI ID 9.
7-0 AID8_BID	This field shows the CAAM Block ID that uses AXI ID 8.

10.13.21 DMA0 AXI ID Enable Register (DMA0_AID_ENB)

The DMA AXI ID Enable register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each CAAM internal connected to it. The assignments are made using the lowest-numbered, available IDs.

NOTE

Note that for backward compatibility the same register is readable at two different addresses.

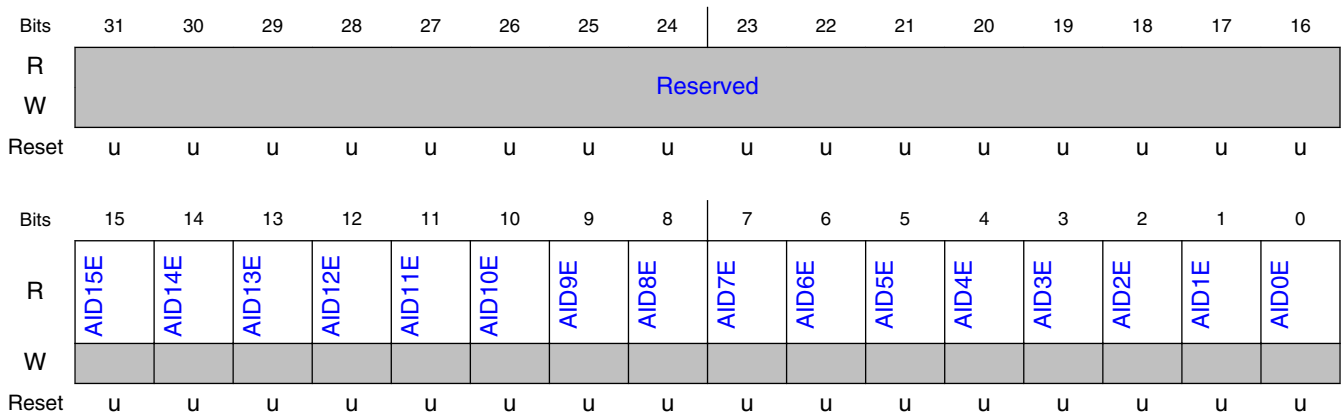
NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.21.1 Offset

Register	Offset	Description
DMA0_AID_ENB	250h	Use of this register alias is deprecated. Instead, use the register alias DMA_X_AID_15_0_EN

10.13.21.2 Diagram



10.13.21.3 Fields

Field	Description
31-16 —	Reserved
15 AID15E	If AID15E=1 then AXI ID 15 is enabled for this DMA engine.
14 AID14E	If AID14E=1 then AXI ID 14 is enabled for this DMA engine.
13 AID13E	If AID13E=1 then AXI ID 13 is enabled for this DMA engine.
12 AID12E	If AID12E=1 then AXI ID 12 is enabled for this DMA engine.
11 AID11E	If AID11E=1 then AXI ID 11 is enabled for this DMA engine.
10 AID10E	If AID10E=1 then AXI ID 10 is enabled for this DMA engine.
9 AID9E	If AID9E=1 then AXI ID 9 is enabled for this DMA engine.
8 AID8E	If AID8E=1 then AXI ID 8 is enabled for this DMA engine.
7 AID7E	If AID7E=1 then AXI ID 7 is enabled for this DMA engine.
6 AID6E	If AID6E=1 then AXI ID 6 is enabled for this DMA engine.

Table continues on the next page...

Field	Description
5 AID5E	If AID5E=1 then AXI ID 5 is enabled for this DMA engine.
4 AID4E	If AID4E=1 then AXI ID 4 is enabled for this DMA engine.
3 AID3E	If AID3E=1 then AXI ID 3 is enabled for this DMA engine.
2 AID2E	If AID2E=1 then AXI ID 2 is enabled for this DMA engine.
1 AID1E	If AID1E=1 then AXI ID 1 is enabled for this DMA engine.
0 AID0E	If AID0E=1 then AXI ID 0 is enabled for this DMA engine.

10.13.22 DMA0 AXI Read Timing Check Register (DMA0_ARD_TC)

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

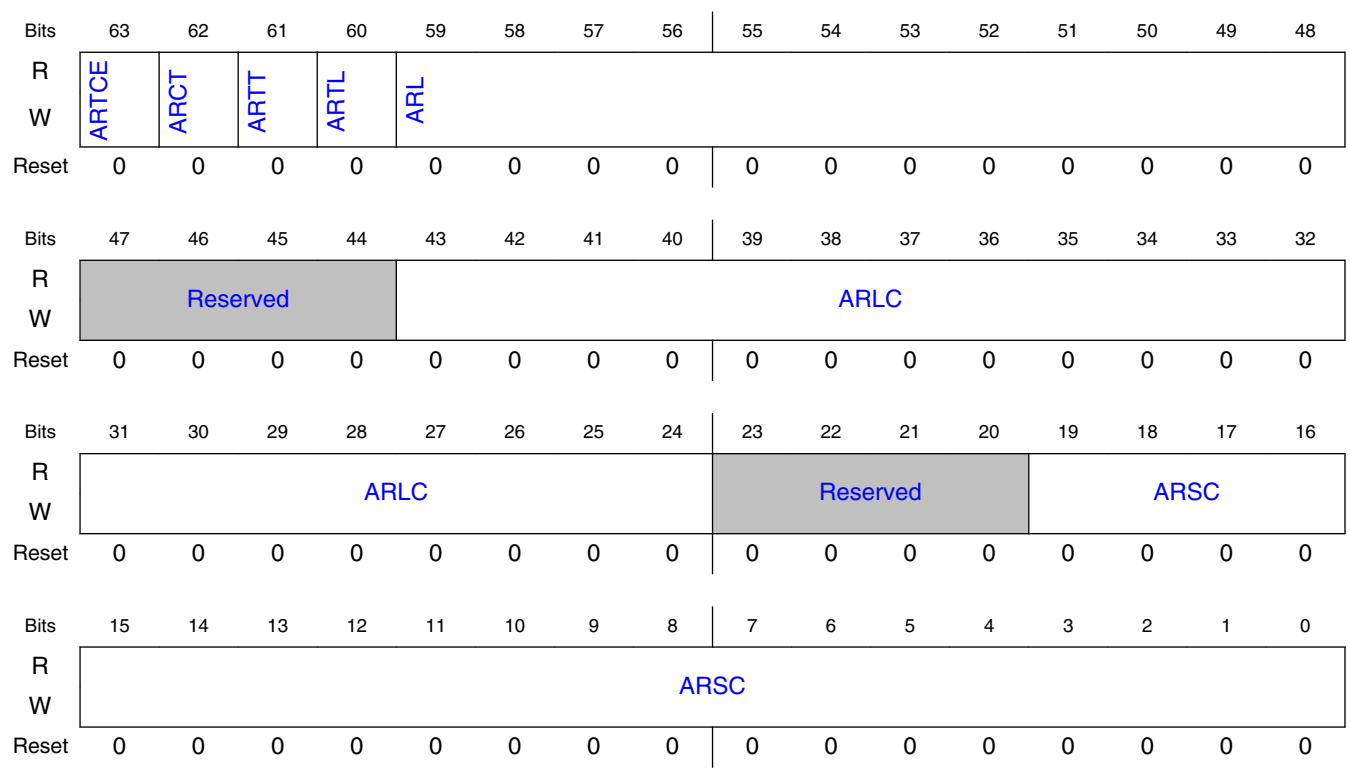
Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMAn_ARD_TC register located in the address range 00260..002EF. Writing to either register affects the

corresponding fields in the other register. But note that some fields in the DMAn_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or DMA_X_ARTC_SC register. The preferred registers are located in the address range 00530..005DF. The use of the DMAn_ARD_TC register located in the address range 00260..002EF is deprecated.

10.13.22.1 Offset

Register	Offset	Description
DMA0_ARD_TC	260h	The addresses of the two halves of the register are unaffected by the endianness configuration.

10.13.22.2 Diagram



10.13.22.3 Fields

Field	Description
63 ARTCE	AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMAn_ARD_TC and SARL in DMAn_SARL and ARL in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC and SARL in DMA_X_ARTC_LC are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only. Note that writing ARTCE in either DMAn_ARD_TC or DMA_X_ARTC_CTL has the same effect.
62 ARCT	AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMAn_ARD_TC, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMAn_SARL and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMAn_ARD_TC and DMAn_SARL or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMA_X_SARL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
61 ARTT	AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The count can optionally be modified to count until the last beat of data instead of the first by setting the ARTL (AXI Read Timer Last) bit. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMAn_ARD_TC register is aliased to bit field ARTT in the DMA_X_ARTC_CTL register, i.e. writing to either ARTT bit field alters the ARTT value in the other register.
60 ARTL	AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement. 0b - A read transaction counter measurement is stopped when the first beat of data arrives 1b - A read transaction counter measurement is stopped when the last beat of data arrives
59-48 ARL	AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL /DMA_X_ARTC_LAT. This field is writeable only when ARTCE=0.Note that bit field ARL in the DMAn_ARD_TC register is aliased to bit field ARL in the DMA_X_ARTC_CTL register, i.e. writing to either ARL bit field alters the ARL value in the other register.
47-44 —	Reserved
43-24 ARLC	AXI Read Late Count. This field is incremented whenever the AXI Read Timer equals or exceeds the AXI Read Limit. AXI read timing checks are suspended when ARLC=FFFFFh. This field is writeable only when ARTCE=0.
23-20 —	Reserved
19-0 ARSC	AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFFh. This field is writeable only when ARTCE=0.

10.13.23 DMA0 Read Timing Check Latency Register (DMA0_ARD_LAT)

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

NOTE

Note that the DMAn_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

10.13.23.1 Offset

Register	Offset
DMA0_ARD_LAT	26Ch

10.13.23.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SARL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SARL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.23.3 Fields

Field	Description
31-0 SARL	Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL. This field is writeable only when ARTCE=0.

10.13.24 DMA0 AXI Write Timing Check Register (DMA0_AWR_TC)

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the write response. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

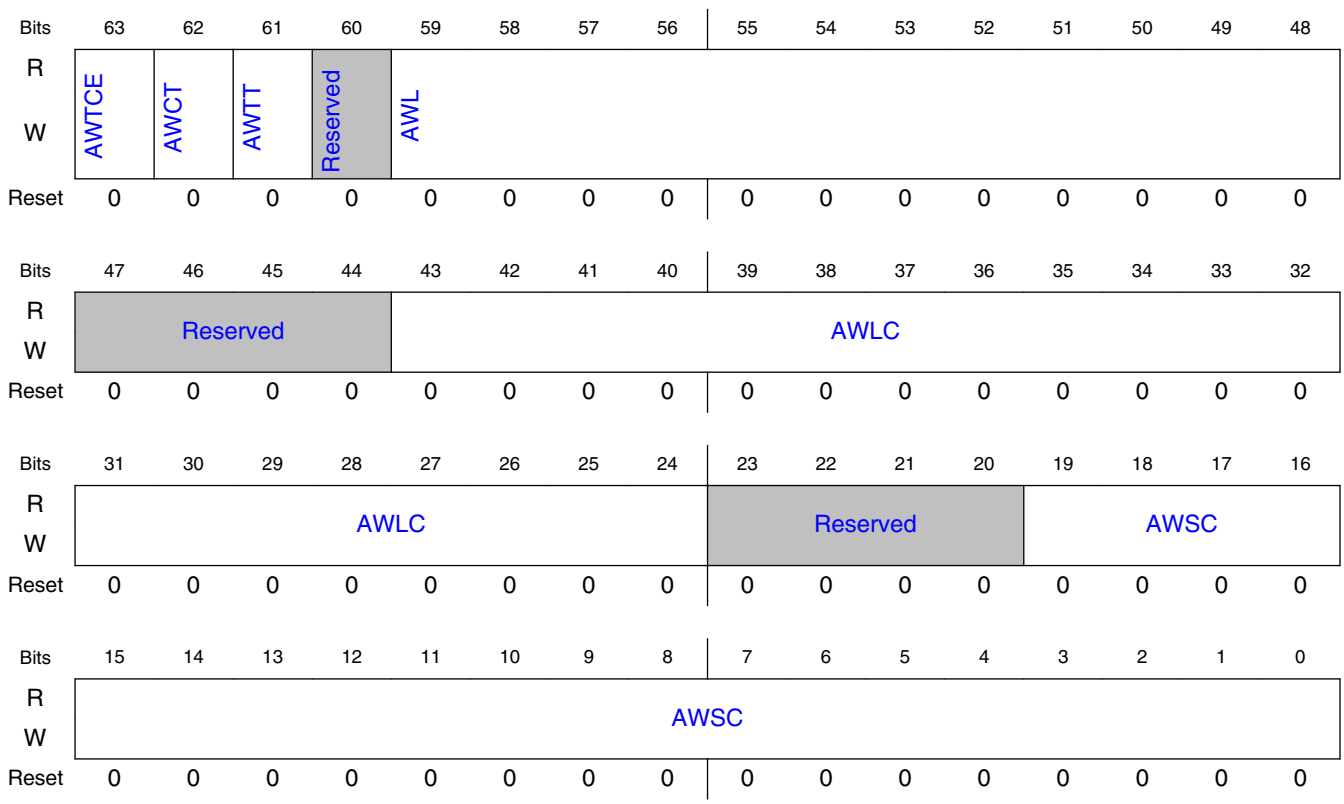
NOTE

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMAn_AWR_TC register located in the address range 00270..002EF. Writing to either register affects the corresponding fields in the other register. But note that some fields in the DMAn_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_TC_SAWL register or DMA_X_AWTC_SC register. The preferred registers are located in the address range 00540..005DF. The use of the DMAn_AWR_TC register located in the address range 00270..002EF is deprecated.

10.13.24.1 Offset

Register	Offset	Description
DMA0_AWR_TC	270h	The addresses of the two halves of the register are unaffected by the endianness configuration.

10.13.24.2 Diagram



10.13.24.3 Fields

Field	Description
63 AWTCE	AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMAn_AWR_TC and SAWL in DMAn_AWL and AWL in DMA_X_AWTC_CTL, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC and SAWL in DMA_X_TC_SAWL are writeable. When AWTCE=1, AXI write timing checks are enabled and these fields are read-only. Note that writing AWTCE in either DMAn_AWR_TC or DMA_X_AWTC_CTL has the same effect.

Table continues on the next page...

Field	Description
62 AWCT	AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMAn_AWR_TC, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC, and SAWL in DMAn_AWL and DMA_X_TC_SAWL, are not cleared when timing checks are enabled and when timing checks resume after writing DMAn_AWR_TC and DMAn_AWL or DMA_X_TC_SAWL, DMA_X_AWTC_SC and DMA_X_AWL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
61 AWTT	AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field AWTT in the DMAn_AWR_TC register is aliased to bit field AWTT in the DMA_X_AWTC_CTL register, i.e. writing to either AWTT bit field alters the AWTT value in the other register.
60 —	Reserved
59-48 AWL	AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. If the latency equals or exceeds the AXI Write Limit, the Write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL /DMA_X_AWTC_LAT. This field is writeable only when ARTCE=0.Note that bit field AWL in the DMAn_AWR_TC register is aliased to bit field AWL in the DMA_X_AWTC_CTL register, i.e. writing to either AWL bit field alters the AWL value in the other register.
47-44 —	Reserved
43-24 AWLC	AXI Write Late Count. This field is incremented whenever the AXI Write Timer equals or exceeds the AXI Write Limit. AXI write timing checks are suspended when AWLC=FFFFFh. This field is writeable only when AWTCE=0.
23-20 —	Reserved
19-0 AWSC	AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFFh. This field is writeable only when AWTCE=0.

10.13.25 DMA0 Write Timing Check Latency Register (DMA0_AWR_LAT)

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

NOTE

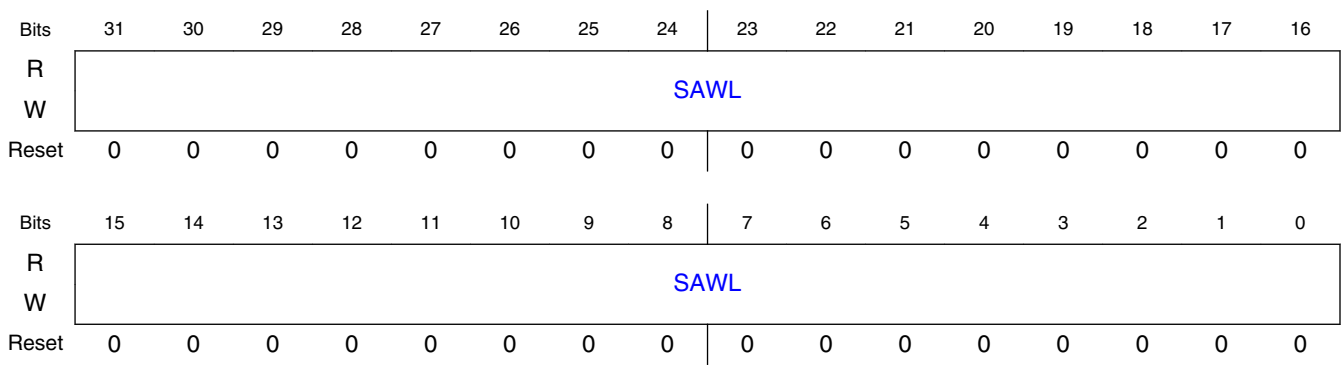
Note that the DMAn_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_AWL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to

consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

10.13.25.1 Offset

Register	Offset
DMA0_AWR_LAT	27Ch

10.13.25.2 Diagram



10.13.25.3 Fields

Field	Description
31-0 SAWL	Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL. This field is writeable only when AWTCE=0.

10.13.26 Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)

The Manufacturing Protection Private Key register (MPPKR) is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to all this by signing a Manufacturing Protection message using the private key stored in the MPPKR. Software running on the SOC then sends this MP attestation message to the OEM's server. The OEM's server can verify that the MP message is genuine by verifying the signature over the signed message using the corresponding ECDSA public key. The public key was previously generated in the OEM's trusted facility by executing the MPPubKey Gen command on an SoC of known provenance, and an integrity-protected copy of the public key was stored on the OEM server for use in verifying Manufacturing Protection attestation messages.

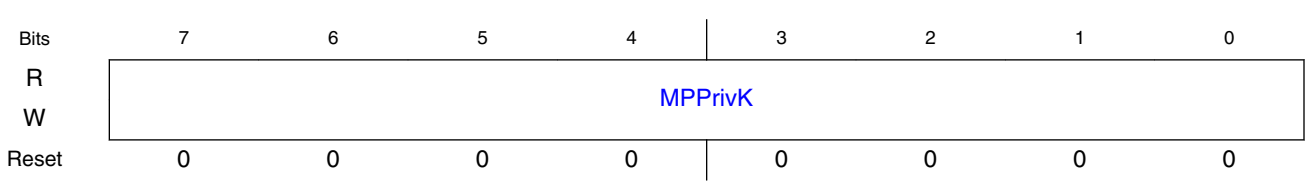
During the SoC's secure boot process, the boot firmware uses the Super Root Key Hash (SRKH) stored in the fuse bank to verify the public key that is then used to verify the public key signature over the device's boot image. Thus successful completion of secure boot verifies that the booted image was signed by a private key whose matching public key was included in the SRKH value. The private key is derived, in part, from the Super Root Key Hash (SRKH), which is used to derive the proper Manufacturing Protection Private Key. The private key in the MPPKR will match the public key held by the OEM's server only if the OEM's trusted manufacturing provisioning software, run at the bulk manufacturing site, signs with all the same source message strings and with the same MPPKR as used by the OEM on the device of known provenance at the OEM's trusted facility. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

10.13.26.1 Offset

For a = 0 to 63:

Register	Offset
MPPKRa	300h + (a × 1h)

10.13.26.2 Diagram



10.13.26.3 Fields

Field	Description
7-0 MPPrivK	MPPrivK. The 512-bit Manufacturing Protection Private Key.

10.13.27 Manufacturing Protection Message Register (MPMR0 - MPMR31)

The Manufacturing Protection Message register is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to this by signing a message using the private key stored in the MPPKR. The message is composed, in part, of the content of the MPMR. The value in the MPMR is written by trusted software. The value normally includes the hash of the public key used to verify the signature over the signed code image. Software running on the SOC then sends this signed message to the OEM’s server. The OEM’s server can confirm that all this information is correct by verifying the signature over the signed message. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

10.13.27.1 Offset

For a = 0 to 31:

Register	Offset
MPMRa	380h + (a × 1h)

10.13.27.2 Diagram

Bits	7	6	5	4	3	2	1	0
R W	MPMSG							
Reset	0	0	0	0	0	0	0	0

10.13.27.3 Fields

Field	Description
7-0 MPMSG	Holds 256 bits of message data that will be prepended to the input data to the MPSIGN operation. When accessed via the register bus this should be treated as a byte array (although it must be accessed as eight 32-bit words).

10.13.28 Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)

The Manufacturing Protection TEST register is used only for hardware verification.

10.13.28.1 Offset

For a = 0 to 31:

Register	Offset
MPTESTRa	3C0h + (a × 1h)

10.13.28.2 Diagram

Bits	7	6	5	4	3	2	1	0
R	TEST_VALUE							
W								
Reset	0	0	0	0	0	0	0	0

10.13.28.3 Fields

Field	Description
7-0 TEST_VALUE	TEST_VALUE. When accessed via the register bus this should be treated as a byte array with the first byte in offset 3C0h (although it must be accessed as eight 32-bit words).(

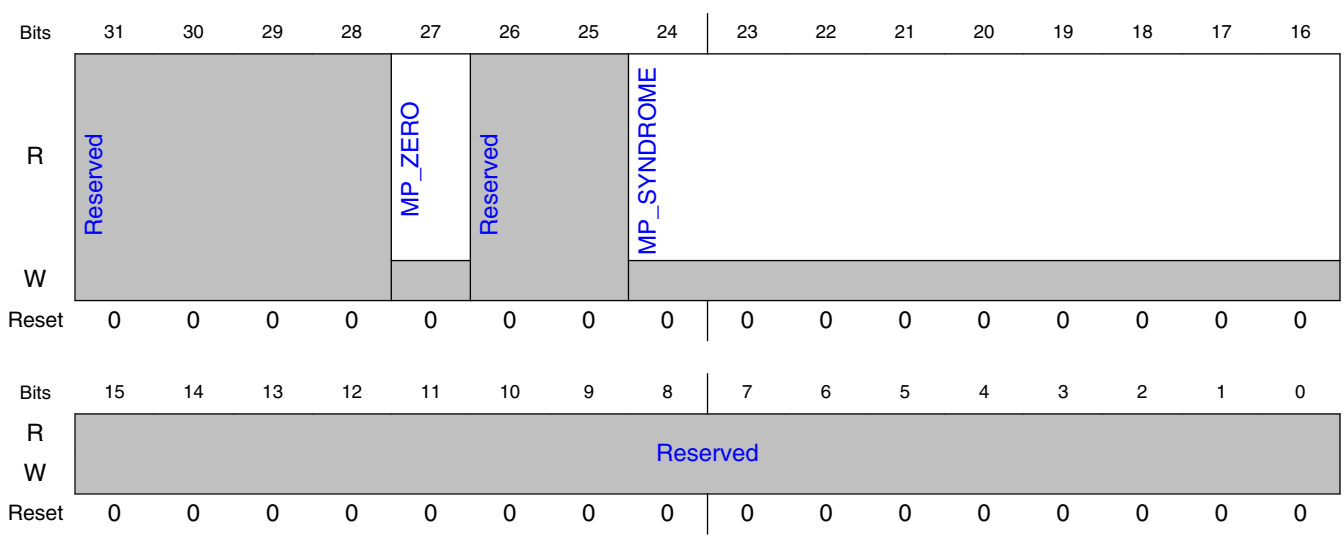
10.13.29 Manufacturing Protection ECC Register (MPECC)

The Manufacturing Protection ECC Register is a read-only register used by software to verify that the Manufacturing Protection Key in the Security Fuse Processor has a valid value. The register will return all zeros if the key is valid.

10.13.29.1 Offset

Register	Offset
MPECC	3F8h

10.13.29.2 Diagram



10.13.29.3 Fields

Field	Description
31-28 —	Reserved
27 MP_ZERO	This bit indicates if the Manufacturing Protection Key that is programmed in the Security Fuse Processor has an all-zero value. 0 - The MP Key in the SFP has a non-zero value. 1 - The MP Key in the SFP is all zeros (unprogrammed).
26-25 —	Reserved
24-16 MP_SYNDROME	This is the syndrome produced by the ECC check on the Manufacturing Protection Key that is programmed in the Security Fuse Processor. An all-zero value indicates that no bits of the key have been corrupted. 000000000 - The MP Key in the SFP passes the ECC check. 000000001-111111111 - The MP Key in the SFP fails the ECC check, and this is the ECC failure syndrome.
15-0 —	Reserved

10.13.30 Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7)

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 Black keys). Since Black Keys are not intended for storage of keys across SOC 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 for use during the new power-on session. The JDKEK is loaded by executing a special descriptor, which can be run in any security mode. (see RNG functional description)

Note that the Secure Mode/Trusted Mode value in JDKEKR is not available when CAAM is in Non-secure Mode because the only possible transitions between Trusted Mode or Secure Mode that lead to Non-secure 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 is read or written via eight 32-bit word addresses. The first byte is in offset 400h.

NOTE

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG. The JDKEKR cannot be read (i.e. reading returns all zeros) or written from the register bus while CAAM is in Secure Mode or Trusted Mode, but JDKEKR can be read and written while CAAM is in Non-secure Mode.

10.13.30.1 Offset

For a = 0 to 7:

Register	Offset
JDKEKRa	400h + (a × 4h)

10.13.30.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	JDKEK															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	JDKEK															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.30.3 Fields

Field	Description
31-0 JDKEK	The 256-bit Job Descriptor Key Encryption Key used to encrypt and decrypt Black Keys.

10.13.31 Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)

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 is read or written via eight 32-bit word addresses. The first byte is in offset 420h.

A new 256-bit secret value is loaded into the TDKEKR from the RNG for use during the new power-on session. The TDKEK is loaded by executing a special descriptor, which can be run in any security mode. (see RNG functional description)

NOTE

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG. The TDKEKR cannot be read (i.e. reading returns all zeros) or written from the register bus while CAAM is in Secure Mode or Trusted Mode, but TDKEKR can be read and written while CAAM is in Non-secure Mode.

10.13.31.1 Offset

For a = 0 to 7:

Register	Offset
TDKEKRa	420h + (a × 4h)

10.13.31.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	TDKEK															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	TDKEK															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.31.3 Fields

Field	Description
31-0 TDKEK	The 256-bit Trusted Descriptor Key Encryption Key used to encrypt and decrypt Black Keys.

10.13.32 Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)

The Trusted Descriptor Signing Key Register contains the TDSK, which is used to generate and verify signatures on Trusted Descriptors. The TDSKR is loaded in the same fashion as the JDKEK. The TDSK is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 440h.

NOTE

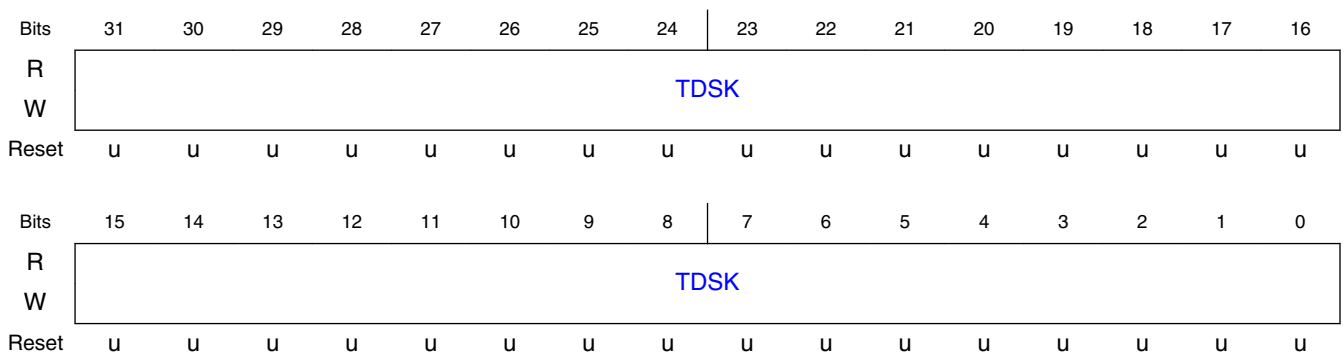
The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.The TDSKR cannot be read (i.e. reading returns all zeros) or written from the register bus while CAAM is in Secure Mode or Trusted Mode, but TDSKR can be read and written while CAAM is in Non-secure Mode.

10.13.32.1 Offset

For a = 0 to 7:

Register	Offset
TDSKRa	440h + (a × 4h)

10.13.32.2 Diagram



10.13.32.3 Fields

Field	Description
31-0 TDSK	The 256-bit Trusted Descriptor Signing Key used to sign and verify Trusted Descriptors.

10.13.33 Secure Key Nonce Register (SKNR)

The Secure Key Nonce Register holds a nonce value that is used for Black Key encryption. To ensure that a nonce is never reused during a power-on session, 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 PKHA E Memory, the AFHA S-Box, the Class 1 Key Register, or the Class 2 Key Register.) The SKNR 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 more than 32 bits, it is accessed over the IP bus as two 32-bit words.

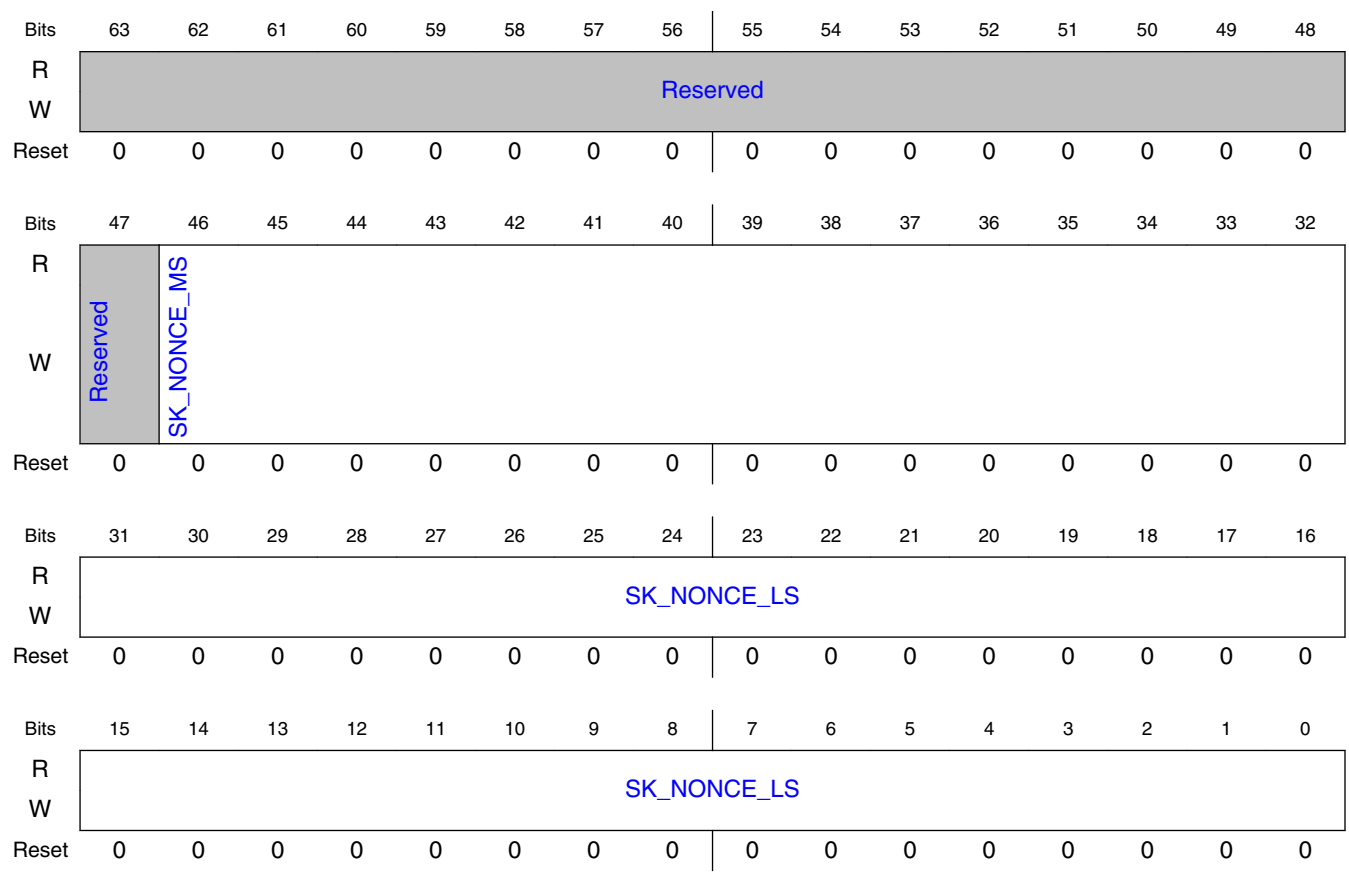
NOTE

This register is writable only when CAAM is in NonSecure mode.

10.13.33.1 Offset

Register	Offset	Description
SKNR	4E0h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.33.2 Diagram



10.13.33.3 Fields

Field	Description
63-47 —	Reserved
46-32 SK_NONCE_MS	Secure Key Nonce - Most Significant Bits. This field holds the 15 most-significant bits of the auto-incrementing secure key nonce field. See the description of the SK_NONCE_LS field for more information.
31-0 SK_NONCE_LS	Secure Key Nonce - Least Significant Bits. This field holds the 32 least-significant bits of the auto-incrementing secure key nonce field. The actual nonce value that is used during AES-CCM encryption of Black Keys consists of the SK_NONCE_MS and the SK_NONCE_LS.

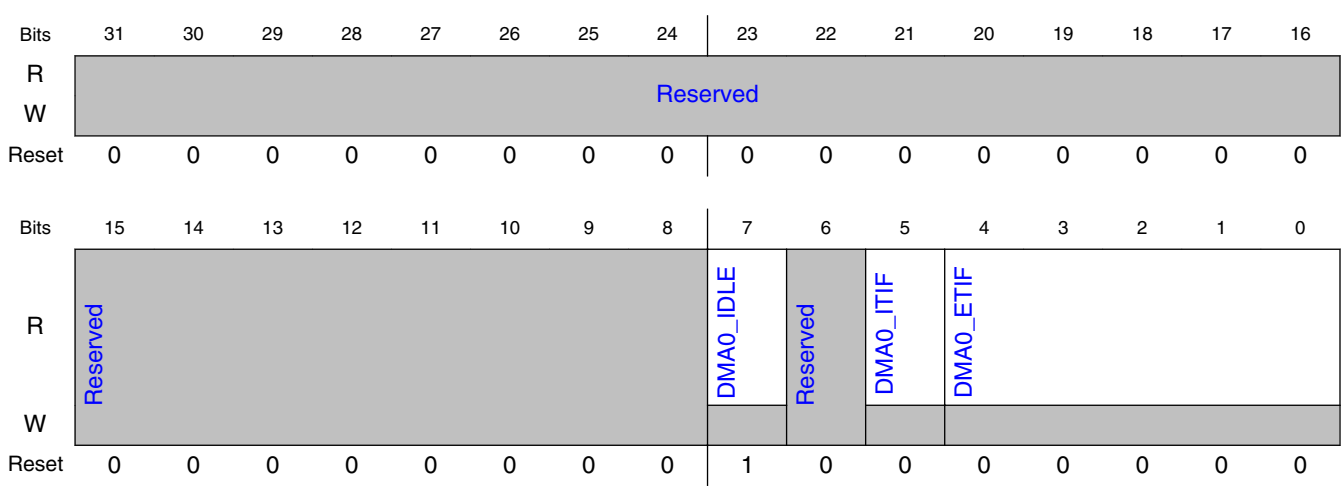
10.13.34 DMA Status Register (DMA_STA)

DMA Status Register

10.13.34.1 Offset

Register	Offset
DMA_STA	50Ch

10.13.34.2 Diagram



10.13.34.3 Fields

Field	Description
31-8 —	Reserved
7 DMA0_IDLE	DMA0 is idle. DMA0's command queue is empty.
6 —	Reserved
5 DMA0_ITIF	DMA0 Internal Transactions in Flight. DMA0_ITIF indicates the number of transactions the DMA0 engine currently has in flight on CAAM's internal bus to Secure Memory.
4-0 DMA0_ETIF	DMA0 External Transactions in Flight. DMA0_ETIF indicates the number of transactions the DMA0 engine currently has in flight on CAAM's external AXI bus.

10.13.35 DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to CAAM internal blocks. These assignments are made via hardwired signals and are SOC-specific. The value of each 8-bit field indicates the internal ID of the CAAM block that will use the AXI ID corresponding to the field. For example, AID2BID=00001000 means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMA_X_ AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMA. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

The CAAM internal block IDs are encoded as follows:

Internal Block ID	Internal Logic Block
00000001b	Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.)
00000100b	Burst Buffer
00001000b	DECO0
All other values are reserved.	

NOTE

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.35.1 Offset

Register	Offset	Description
DMA_X_AID_7_4_MAP	510h	Mapping for DMA AXI IDs 7 ... 4

10.13.35.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID7_BID								AID6_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID5_BID								AID4_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.35.3 Fields

Field	Description
31-24 AID7_BID	This field shows the CAAM Block ID that uses AXI ID 7.
23-16 AID6_BID	This field shows the CAAM Block ID that uses AXI ID 6.
15-8 AID5_BID	This field shows the CAAM Block ID that uses AXI ID 5.

Table continues on the next page...

Field	Description
7-0 AID4_BID	This field shows the CAAM Block ID that uses AXI ID 4.

10.13.36 DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.36.1 Offset

Register	Offset	Description
DMA_X_AID_3_0_MAP	514h	Mapping for DMA AXI IDs 3 ... 0

10.13.36.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID3_BID								AID2_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID1_BID								AID0_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.36.3 Fields

Field	Description
31-24 AID3_BID	This field shows the CAAM Block ID that uses AXI ID 3.
23-16 AID2_BID	This field shows the CAAM Block ID that uses AXI ID 2.
15-8 AID1_BID	This field shows the CAAM Block ID that uses AXI ID 1.
7-0 AID0_BID	This field shows the CAAM Block ID that uses AXI ID 0.

10.13.37 DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.37.1 Offset

Register	Offset	Description
DMA_X_AID_15_12_MAP	518h	Mapping for DMA AXI IDs 15 ... 12

10.13.37.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID15_BID								AID14_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID13_BID								AID12_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.37.3 Fields

Field	Description
31-24 AID15_BID	This field shows the CAAM Block ID that uses AXI ID 15.
23-16 AID14_BID	This field shows the CAAM Block ID that uses AXI ID 14.
15-8 AID13_BID	This field shows the CAAM Block ID that uses AXI ID 13.
7-0 AID12_BID	This field shows the CAAM Block ID that uses AXI ID 12.

10.13.38 DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to CAAM internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.38.1 Offset

Register	Offset	Description
DMA_X_AID_11_8_MAP	51Ch	Mapping for DMA AXI IDs 11 ... 8

10.13.38.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AID11_BID								AID10_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AID9_BID								AID8_BID							
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

10.13.38.3 Fields

Field	Description
31-24 AID11_BID	This field shows the CAAM Block ID that uses AXI ID 11.
23-16 AID10_BID	This field shows the CAAM Block ID that uses AXI ID 10.
15-8 AID9_BID	This field shows the CAAM Block ID that uses AXI ID 9.
7-0 AID8_BID	This field shows the CAAM Block ID that uses AXI ID 8.

10.13.39 DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN)

The DMA_X_AID_15_0_EN register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each CAAM internal connected to it. The assignments are made using the lowest-numbered, available IDs.

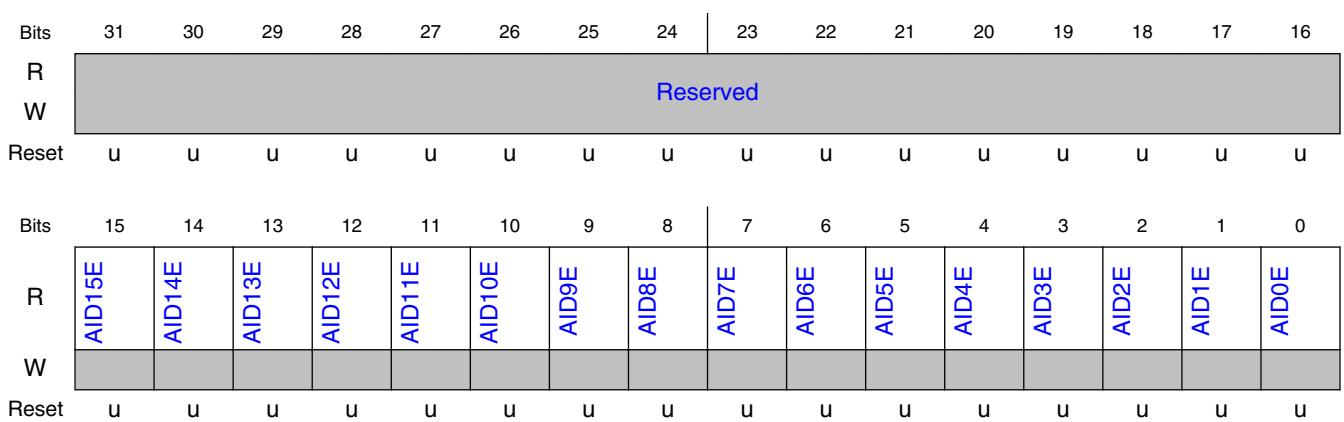
NOTE

Note that for backward compatibility the same information is readable at two different addresses, 250h and 524h. The 250h address is deprecated.

10.13.39.1 Offset

Register	Offset	Description
DMA_X_AID_15_0_EN	524h	For new software DMA_X_AID_15_0_EN (address 510h) should be used rather than DMA_0_AID_ENB (address 250h). **NOTE:** The values read from this register are determined by hardwired inputs to CAAM and are SoC-specific.

10.13.39.2 Diagram



10.13.39.3 Fields

Field	Description
31-16	Reserved

Table continues on the next page...

Field	Description
—	
15 AID15E	If AID15E=1 then AXI ID 15 is enabled for this DMA engine.
14 AID14E	If AID14E=1 then AXI ID 14 is enabled for this DMA engine.
13 AID13E	If AID13E=1 then AXI ID 13 is enabled for this DMA engine.
12 AID12E	If AID12E=1 then AXI ID 12 is enabled for this DMA engine.
11 AID11E	If AID11E=1 then AXI ID 11 is enabled for this DMA engine.
10 AID10E	If AID10E=1 then AXI ID 10 is enabled for this DMA engine.
9 AID9E	If AID9E=1 then AXI ID 9 is enabled for this DMA engine.
8 AID8E	If AID8E=1 then AXI ID 8 is enabled for this DMA engine.
7 AID7E	If AID7E=1 then AXI ID 7 is enabled for this DMA engine.
6 AID6E	If AID6E=1 then AXI ID 6 is enabled for this DMA engine.
5 AID5E	If AID5E=1 then AXI ID 5 is enabled for this DMA engine.
4 AID4E	If AID4E=1 then AXI ID 4 is enabled for this DMA engine.
3 AID3E	If AID3E=1 then AXI ID 3 is enabled for this DMA engine.
2 AID2E	If AID2E=1 then AXI ID 2 is enabled for this DMA engine.
1 AID1E	If AID1E=1 then AXI ID 1 is enabled for this DMA engine.
0 AID0E	If AID0E=1 then AXI ID 0 is enabled for this DMA engine.

10.13.40 DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the DMA_X AXI Read Latency Register or DMA_X Read Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMAn_ARD_TC register located in the address range 00260..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMAn_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or the DMA_X_ARTC_SC register.

10.13.40.1 Offset

Register	Offset
DMA_X_ARTC_CTL	530h

10.13.40.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ARTCE	ARCT	ARTT	ARTL	ARL											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				ART											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.40.3 Fields

Field	Description
31 ARTCE	AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMA_X_ARTC and SARL in DMAn_ARD_LAT and ARL in DMA_X_ARTC_CTL, ARLC in DMA0_ARTC_LC, ARSC in DMA0ARTC_SC and SARL in DMA0ARL_LAT are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only. **NOTE:** Note that writing ARTCE in either DMA_X_ARTC_B or DMA_X_ARTC_CTL has the same effect.
30 ARCT	AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMAn_ARD_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMAn_ARD_TC and DMAn_ARD_LAT or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMAn_ARD_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
29 ARTT	AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMA_X_ARTC_CTL register is aliased to bit field ARTT in the DMAn_ARD_TC register, i.e. writing to either ARTT bit field alters the ARTT value in the other register.
28 ARTL	AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement. 0b - A read transaction counter measurement is stopped when the first beat of data arrives 1b - A read transaction counter measurement is stopped when the last beat of data arrives
27-16 ARL	AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_ARD_LAT/DMA_X_ARTC_LAT. This field is

Table continues on the next page...

Field	Description
	writeable only when ARTCE=0. Note that bit field ARL in the DMA_X_ARTC_CTL register is aliased to bit field ARL in the DMAn_ARD_TC register, i.e. writing to either ARL bit field alters the ARL value in the other register.
15-12 —	Reserved
11-0 ART	AXI Read Timer. The number of AXI clock cycles from the latest external AXI read address transaction initiated by this DMA to the beginning of the corresponding read data transaction. This field is writeable only when ARTCE=0.

10.13.41 DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the ARTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

NOTE

Note that the DMA_X_ARTC_LC register provides functionality similar to the AXI Read Timing Late Check fields in the DMAn_ARD_TC register located in the address range 00260..002EF, but the fields have been rearranged. Usage of the DMAn_ARD_TC register is deprecated.

10.13.41.1 Offset

Register	Offset
DMA_X_ARTC_LC	534h

10.13.41.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved												ARLC			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ARLC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.41.3 Fields

Field	Description
31-20 —	Reserved
19-0 ARLC	AXI Read Late Count. This field is incremented each time that the ART field exceeds the ARL field in the DMA_X AXI Read Timing Check Control Register. AXI read timing checks are suspended when ARLC=FFFFFh. Note that this field is an alias of the ARLC field in the DMAn_ARD_TC Register. Reading or writing the ARLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMAn_ARD_TC/ DMA_X_ARTC_TC[ARTCE]=0, the ARLC bit field in DMAn_ARD_TC and the DMA_X_ARTC_LC register are writeable. When ARTCE=1, AXI read timing checks are enabled and the ARLC bit fields are read-only.

10.13.42 DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or

- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

Note that the ARSC field in the DMA_X_ARTC_SC register located in the address range 00530..005DF provides functionality equivalent to the ARSC field in the DMAn_ARD_TC register located in the address range 00260..002EF. Writing to the ARSC bit field in either register affects the ARSC bit field in the other register.

10.13.42.1 Offset

Register	Offset
DMA_X_ARTC_SC	538h

10.13.42.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved												ARSC			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ARSC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.42.3 Fields

Field	Description
31-20 —	Reserved

Table continues on the next page...

Field	Description
19-0 ARSC	AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFFh. This field is writeable only when ARTCE=0.

10.13.43 DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

NOTE

Note that the DMA_X_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

10.13.43.1 Offset

Register	Offset
DMA_X_ARTC_LAT	53Ch

10.13.43.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SARL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SARL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.43.3 Fields

Field	Description
31-0 SARL	Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL. This field is writeable only when ARTCE=0.

10.13.44 DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the DMA_X AXI Write Latency Register or DMA_X Write Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

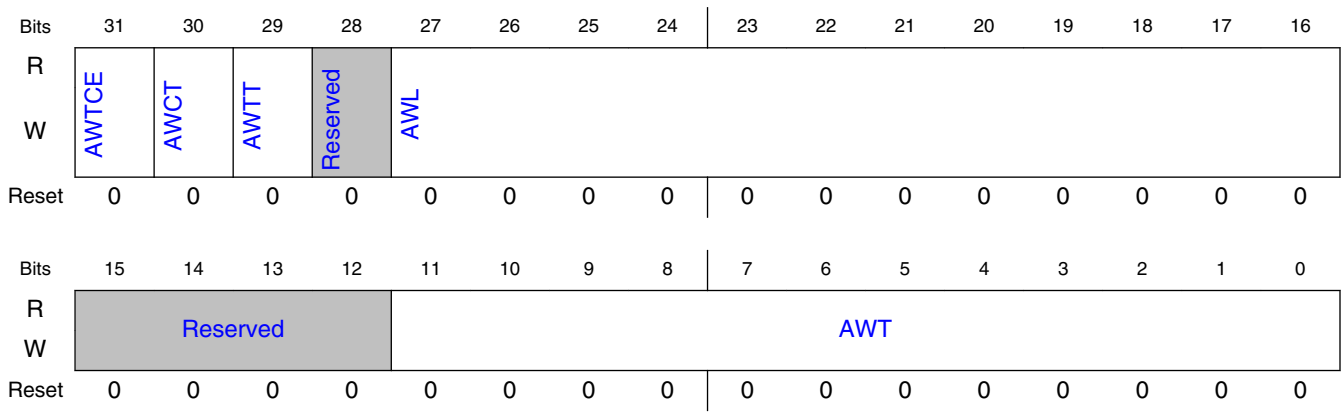
NOTE

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMAn_AWR_TC register located in the address range 00270..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMAn_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_AWTC_LC register or the DMA_X_AWTC_SC register.

10.13.44.1 Offset

Register	Offset
DMA_X_AWTC_CTL	540h

10.13.44.2 Diagram



10.13.44.3 Fields

Field	Description
31 AWTCE	AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMA_X_AWTC and SAWL in DMAn_AWR_LAT and AWL in DMA_X_AWTC_CTL, AWLC in DMA0_AWTC_LC, AWSC in DMA0AWTC_SC and SARL in DMA0AWL_LAT are writeable. When AWTCE=1, AXI Write timing checks are enabled and these fields are Write-only. **NOTE:** Note that writing AWTCE in either DMAn_AWR_TC or DMA_X_AWTC_CTL has the same effect.
30 AWCT	AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMA_X_AWTC_CTL, AWLC in DMA_X_AWTC_LC, ARSC in DMA_X_WRTC_SC, and SAWL in DMAn_AWR_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMAn_AWR_TC and DMAn_AWR_LAT or DMA_X_AWTC_LC, DMA_X_AWTC_SC and DMA_X_AWTC_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
29 AWTT	AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops

Table continues on the next page...

Field	Description
	at the terminal value until the next timing check starts. Note that bit field AWTT in the DMA_X_AWTC_CTL register is aliased to bit field AWTT in the DMAn_AWR_TC register, i.e. writing to either AWTT bit field alters the AWTT value in the other register.
28 —	Reserved
27-16 AWL	AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding write data transaction. If the latency equals or exceeds the AXI Write Limit, the write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWR_LAT/DMA_X_ARTC_LAT. This field is writeable only when AWTCE=0. Note that bit field AWL in the DMA_X_AWTC_CTL register is aliased to bit field AWL in the DMAn_AWR_TC register, i.e. writing to either AWL bit field alters the AWL value in the other register.
15-12 —	Reserved
11-0 AWT	AXI Write Timer. The number of AXI clock cycles from the latest external AXI write address transaction initiated by this DMA to the beginning of the corresponding write data transaction. This field is writeable only when AWTCE=0.

10.13.45 DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the AWTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is Write

NOTE

Note that the DMA_X_AWTC_LC register provides functionality similar to the AXI Write Timing Late Check fields in the DMAn_AWR_TC register located in the address range 00270..002EF, but the fields have been rearranged. Usage of the DMAn_AWR_TC register is deprecated.

10.13.45.1 Offset

Register	Offset
DMA_X_AWTC_LC	544h

10.13.45.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved												AWLC			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	AWLC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.45.3 Fields

Field	Description
31-20 —	Reserved
19-0 AWLC	AXI Write Late Count. This field is incremented each time that the AWT field exceeds the AWL field in the DMA_X AXI Write Timing Check Control Register. AXI Write timing checks are suspended when AWLC=FFFFFh. Note that this field is an alias of the AWLC field in the DMAn_AWR_TC Register. Reading or writing the AWLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMAn_AWR_TC/ DMA_X_AWTC_TC[AWTCE]=0, the AWLC bit field in DMAn_AWR_TC and the DMA_X_AWTC_LC register are writeable. When AWTCE=1, AXI write timing checks are enabled and the AWLC bit fields are read-only.

10.13.46 DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

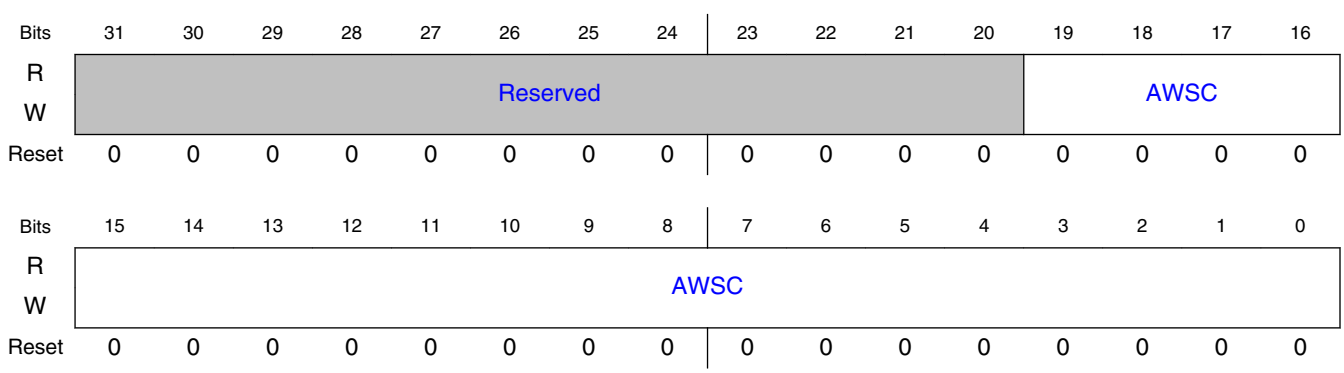
NOTE

Note that the AWSC field in the DMA_X_AWTC_SC register located in the address range 00530..005DF provides functionality equivalent to the AWSC field in the DMAn_AWR_TC register located in the address range 00260..002EF. Writing to the AWSC bit field in either register affects the AWSC bit field in the other register.

10.13.46.1 Offset

Register	Offset
DMA_X_AWTC_SC	548h

10.13.46.2 Diagram



10.13.46.3 Fields

Field	Description
31-20 —	Reserved
19-0 AWSC	AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFFh. This field is writeable only when AWTCE=0.

10.13.47 DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

NOTE

Note that the DMA_X_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_AWL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

10.13.47.1 Offset

Register	Offset
DMA_X_AWTC_LAT	54Ch

10.13.47.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SAWL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SAWL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.47.3 Fields

Field	Description
31-0 SAWL	Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL. This field is writeable only when AWTCE=0.

10.13.48 RNG TRNG Miscellaneous Control Register (RTMCTL)

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.

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.

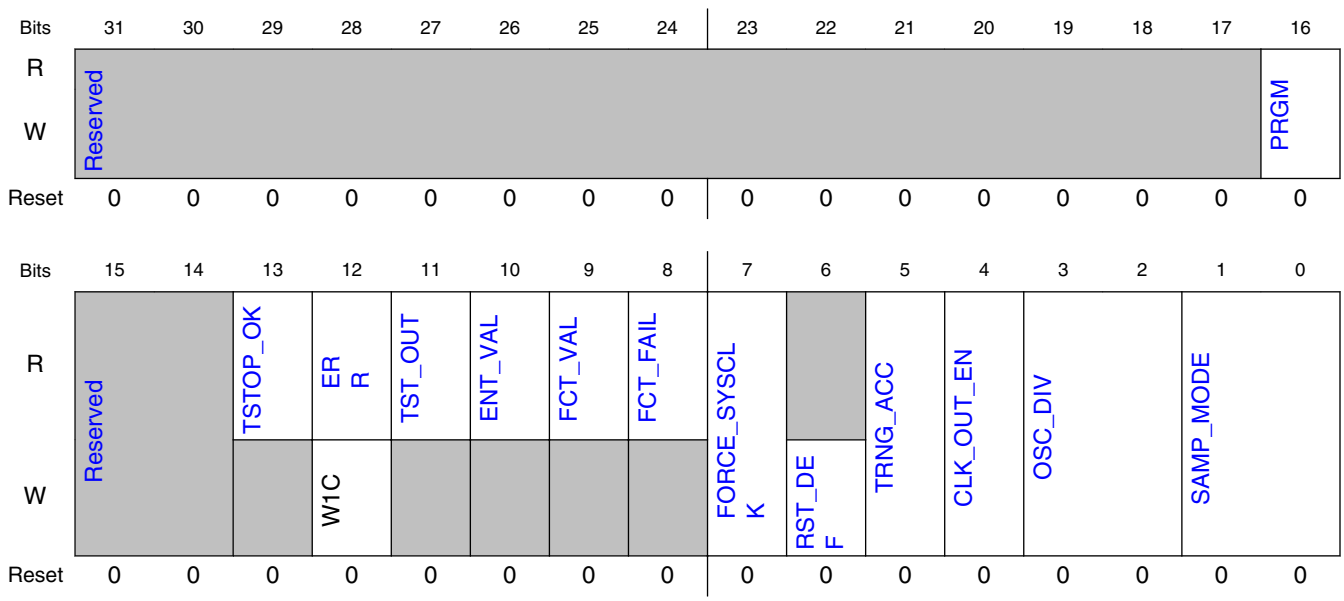
NOTE

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.

10.13.48.1 Offset

Register	Offset
RTMCTL	600h

10.13.48.2 Diagram



10.13.48.3 Fields

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. Note that different RNG registers are accessible at the same address depending on whether PRGM is set to 1 or 0. This is noted in the RNG register descriptions. **NOTE:** If PRGM is set to 1, then TRNG_ACC should also be set to 1.
15-14 —	Reserved
13 TSTOP_OK	TRNG_OK_TO_STOP. Software should check that this bit is a 1 before transitioning CAAM to low power mode (CAAM clock stopped). CAAM turns on the TRNG free-running ring oscillator whenever new entropy is being generated and turns off the ring oscillator when entropy generation is complete. If the

Table continues on the next page...

Field	Description
	CAAM clock is stopped while the TRNG ring oscillator is running, the oscillator will continue running even though the CAAM clock is stopped. TSTOP_OK is asserted when the TRNG ring oscillator is not running. and therefore it is OK to stop the CAAM clock.
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 RTENT15 is read. (RTENT0 through RTENT14 should be read before reading RTENT15).
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.
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 RTENT 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. **NOTE:** If PRGM is set to 1, then TRNG_ACC should also be set to 1.
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. 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. 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
1-0 SAMP_MODE	Sample Mode. Determines the method of sampling the ring oscillator while generating the Entropy value: 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 01 by writing the RST_DEF bit to 1. 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.

10.13.49 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.

NOTE

Reset occurs at POR, and when RTMCTL[RST_DEF] is written to 1.

10.13.49.1 Offset

Register	Offset
RTSCMISC	604h

10.13.49.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved												RTY_CNT			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved								LRUN_MAX							
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

10.13.49.3 Fields

Field	Description
31-20	Reserved

Table continues on the next page...

Field	Description
—	
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 34 by writing the RTMCTL[RST_DEF] bit to 1.

10.13.50 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.

10.13.50.1 Offset

Register	Offset
RTPKRRNG	608h

10.13.50.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PKR_RNG															
Reset	0	0	0	0	1	0	0	1	1	0	1	0	0	0	1	1

10.13.50.3 Fields

Field	Description
31-16 —	Reserved
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.

10.13.51 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

This offset (060Ch) is used as RTPKRMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register.

10.13.51.1 Offset

Register	Offset	Description
RTPKRMAX	60Ch	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.51.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved								PKR_MAX							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PKR_MAX															
Reset	0	1	1	0	1	0	0	1	0	0	1	0	0	0	0	0

10.13.51.3 Fields

Field	Description
31-24 —	Reserved
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.

10.13.52 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

This offset (060Ch) is used as RTPKRMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register, as described here.

10.13.52.1 Offset

Register	Offset	Description
RTPKRSQ	60Ch	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.52.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved								PKR_SQ							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PKR_SQ															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.52.3 Fields

Field	Description
31-24 —	Reserved
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.

10.13.53 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 be taken during each TRNG Entropy generation (SAMP_SIZE).

10.13.53.1 Offset

Register	Offset
RTSDCTL	610h

10.13.53.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ENT_DLY															
Reset	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SAMP_SIZE															
Reset	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0

10.13.53.3 Fields

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 00C80h (decimal 3200) by writing the RTMCTL[RST_DEF] bit to 1.
15-0 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.

10.13.54 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

This address (0614h) is used as RTSBLIM only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTTOTSAM readback register.

10.13.54.1 Offset

Register	Offset	Description
RTSBLIM	614h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.54.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved								SB_LIM							
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

10.13.54.3 Fields

Field	Description
31-10 —	Reserved
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.

10.13.55 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.

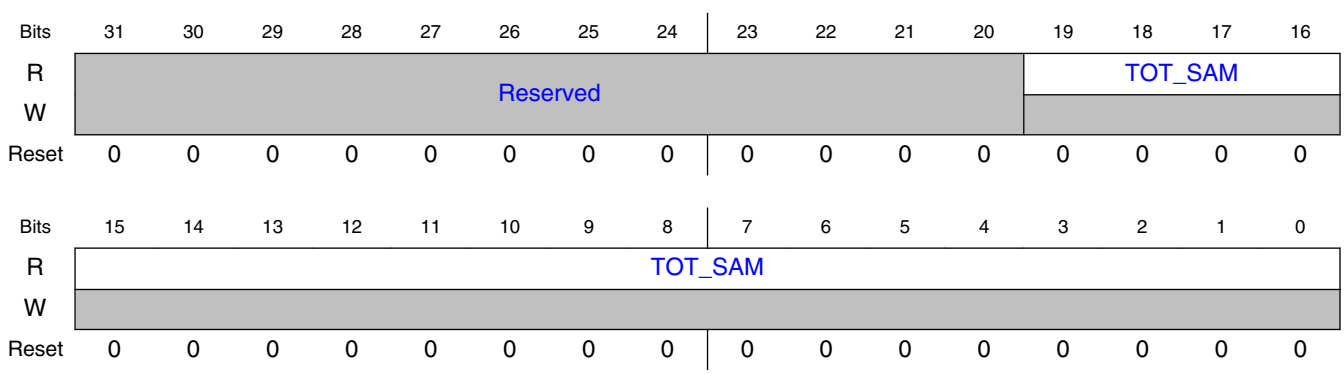
NOTE

This offset (0614h) is used as RTSBLIM if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTTOTSAM readback register, as described here.

10.13.55.1 Offset

Register	Offset	Description
RTTOTSAM	614h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.55.2 Diagram



10.13.55.3 Fields

Field	Description
31-20 —	Reserved

Table continues on the next page...

Field	Description
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.

10.13.56 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.

10.13.56.1 Offset

Register	Offset
RTFRQMIN	618h

10.13.56.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved										FRQ_MIN					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	FRQ_MIN															
Reset	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0

10.13.56.3 Fields

Field	Description
31-22 —	Reserved
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 000190h by writing the RTMCTL[RST_DEF] bit to 1.

10.13.57 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 and OSC2_CTL[TRNG_ENT_CTL] != 10b.

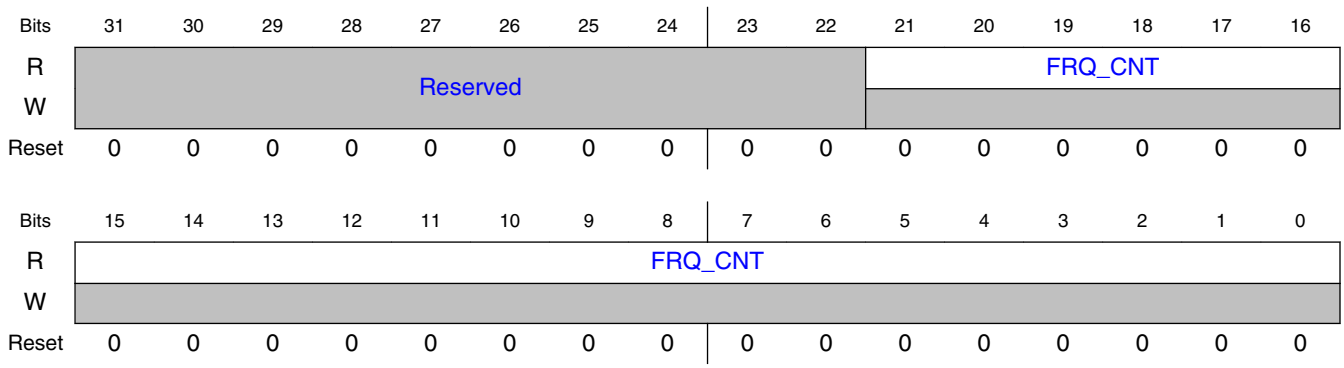
NOTE

This offset (061Ch) is used as RTFRQMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTFRQCNT readback register, as described here.

10.13.57.1 Offset

Register	Offset	Description
RTFRQCNT	61Ch	The RNG TRNG Frequency Count register is accessible at this address when RTMCTL[PRGM] = 0

10.13.57.2 Diagram



10.13.57.3 Fields

Field	Description
31-22 —	Reserved
21-0 FRQ_CNT	Frequency Count. If RTMCTL[TRNG_ACC] = 1, reads a sample frequency count taken during entropy generation. Requires RTMCTL[PRGM] = 0. The value read from FRQ_CNT is valid only if RTMCTL[FCT_VAL] = 1.

10.13.58 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.

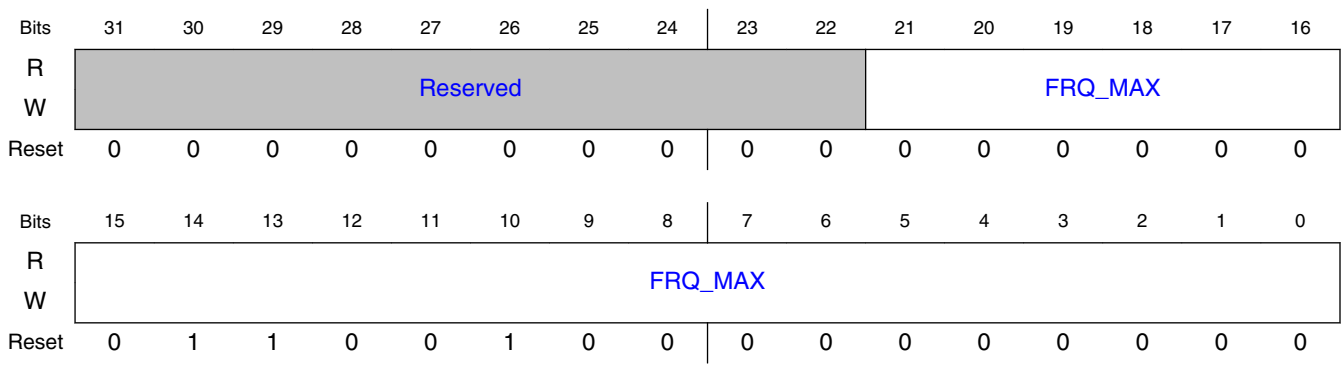
NOTE

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.

10.13.58.1 Offset

Register	Offset	Description
RTFRQMAX	61Ch	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.58.2 Diagram



10.13.58.3 Fields

Field	Description
31-22 —	Reserved
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 00190h 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.

10.13.59 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

This offset (0620h) is used as RTSCML if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register, as described here.

10.13.59.1 Offset

Register	Offset	Description
RTSCMC	620h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.59.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MONO_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.59.3 Fields

Field	Description
31-16 —	Reserved
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.

10.13.60 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/zeroes generated must be less than the programmed maximum value, and the number of ones/zeroes 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

This offset (0620h) is used as RTSCML only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register.

10.13.60.1 Offset

Register	Offset	Description
RTSCML	620h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.60.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MONO_RNG															
Reset	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MONO_MAX															
Reset	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0

10.13.60.3 Fields

Field	Description
31-16 MONO_RNG	Monobit Range. The number of ones/zeroes 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.

Table continues on the next page...

Field	Description
15-0 MONO_MAX	Monobit Maximum Limit. Defines the maximum allowable count taken during entropy generation. The number of ones/zeroes 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.

10.13.61 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).

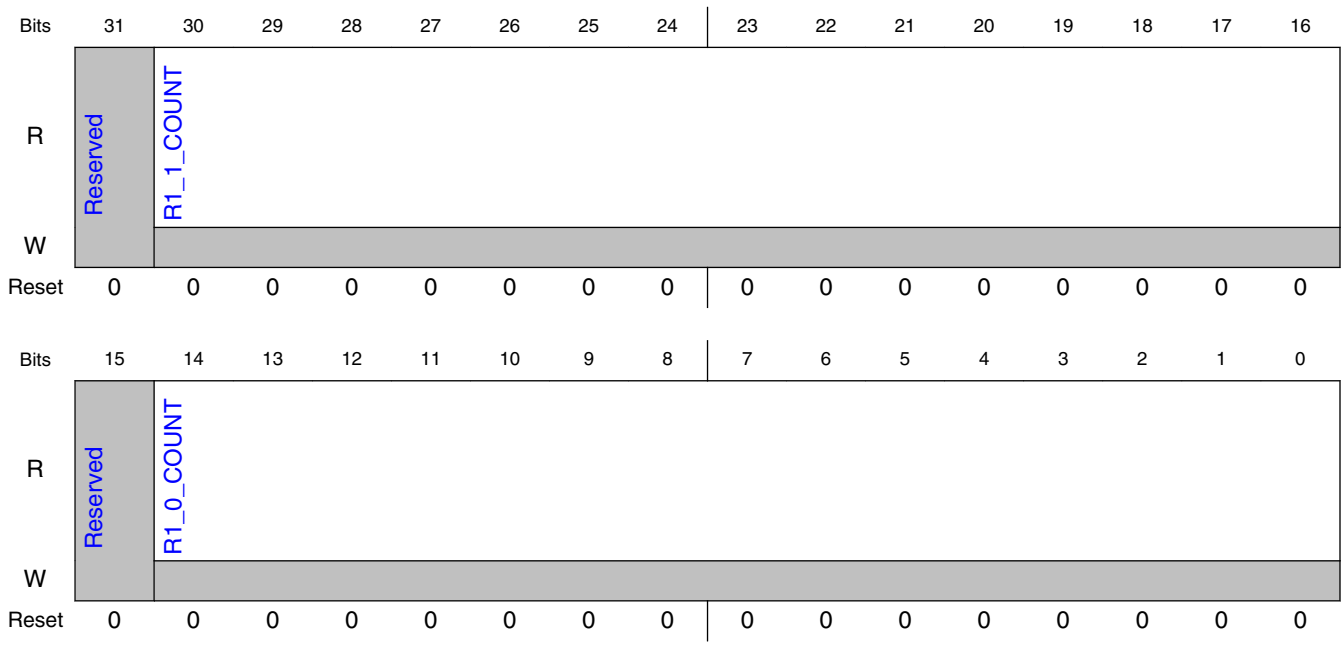
NOTE

This offset (0624h) is used as RTSCR1L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR1C readback register, as described here.

10.13.61.1 Offset

Register	Offset	Description
RTSCR1C	624h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.61.2 Diagram



10.13.61.3 Fields

Field	Description
31 —	Reserved
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
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.

10.13.62 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

This address (0624h) is used as RTSCR1L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR1C readback register.

10.13.62.1 Offset

Register	Offset	Description
RTSCR1L	624h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.62.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved	RUN1_RNG														
W																
Reset	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved	RUN1_MAX														
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	1	0	1	0	1

10.13.62.3 Fields

Field	Description
31 —	Reserved

Table continues on the next page...

Field	Description
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
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.

10.13.63 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).

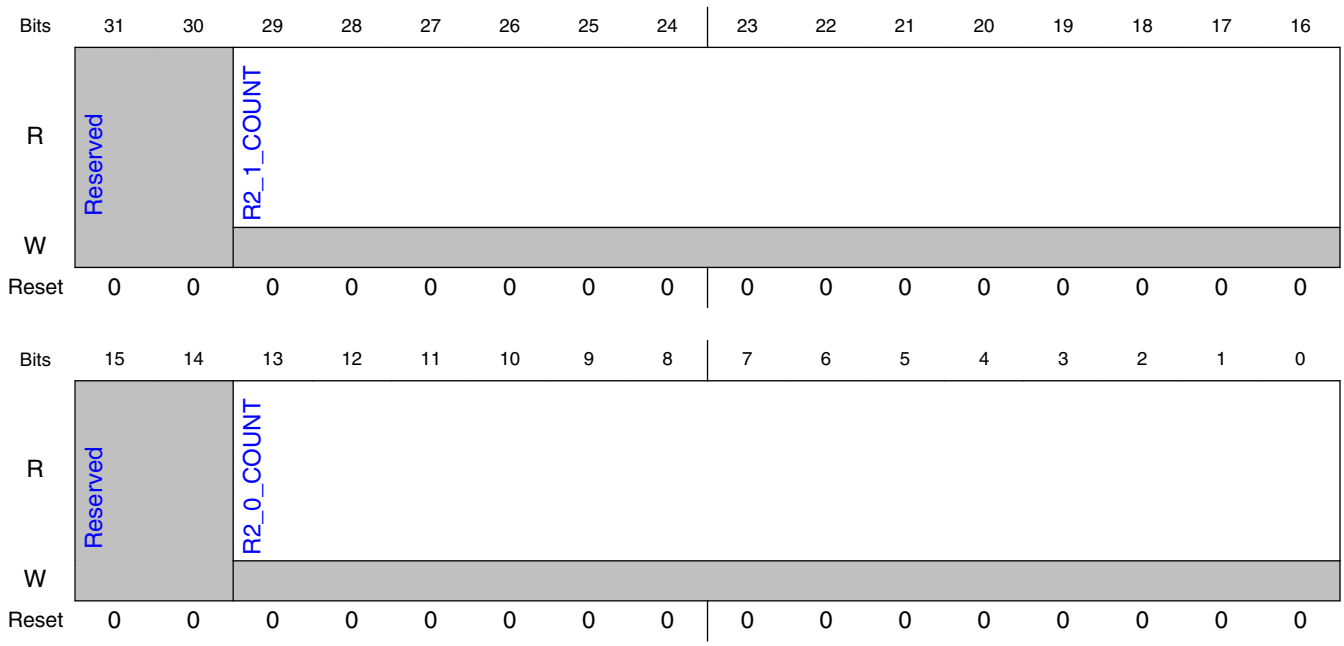
NOTE

This offset (0628h) is used as RTSCR2L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR2C readback register, as described here.

10.13.63.1 Offset

Register	Offset	Description
RTSCR2C	628h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.63.2 Diagram



10.13.63.3 Fields

Field	Description
31-30 —	Reserved
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
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.

10.13.64 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

This address (0628h) is used as RTSCR2L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR2C readback register.

10.13.64.1 Offset

Register	Offset	Description
RTSCR2L	628h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.64.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RUN2_RNG													
W																
Reset	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RUN2_MAX													
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0

10.13.64.3 Fields

Field	Description
31-30 —	Reserved

Table continues on the next page...

Field	Description
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
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.

10.13.65 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).

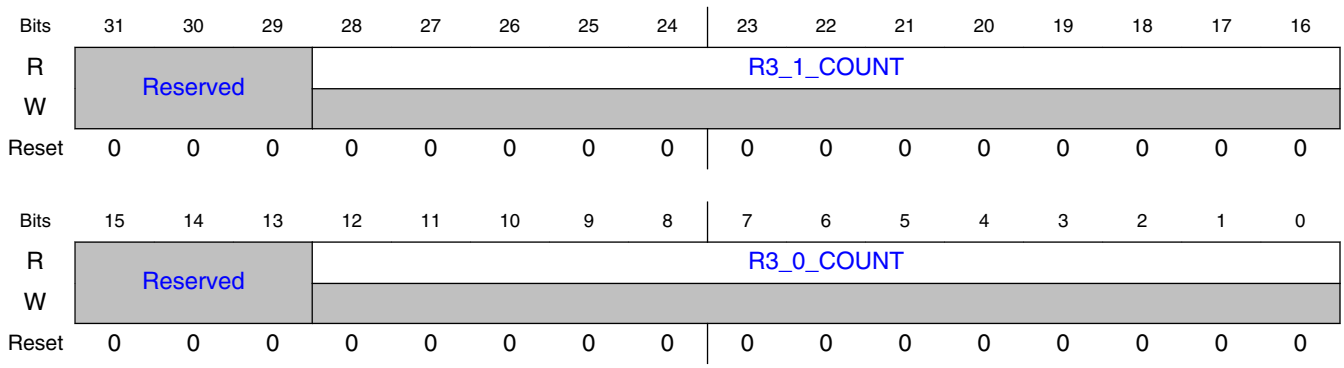
NOTE

This offset (062Ch) is used as RTSCR3L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR3C readback register, as described here.

10.13.65.1 Offset

Register	Offset	Description
RTSCR3C	62Ch	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.65.2 Diagram



10.13.65.3 Fields

Field	Description
31-29 —	Reserved
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
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.

10.13.66 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

This address (062Ch) is used as RTSCR3L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR3C readback register.

10.13.66.1 Offset

Register	Offset	Description
RTSCR3L	62Ch	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.66.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved			RUN3_RNG												
Reset	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved			RUN3_MAX												
Reset	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1

10.13.66.3 Fields

Field	Description
31-29 —	Reserved
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
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.

10.13.67 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

This offset (0630h) is used as RTSCR4L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR4C readback register, as described here.

10.13.67.1 Offset

Register	Offset	Description
RTSCR4C	630h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.67.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				R4_1_COUNT											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				R4_0_COUNT											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.67.3 Fields

Field	Description
31-28 —	Reserved
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
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.

10.13.68 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

This address (0630h) is used as RTSCR4L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR4C readback register.

10.13.68.1 Offset

Register	Offset	Description
RTSCR4L	630h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.68.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved				RUN4_RNG											
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				RUN4_MAX											
Reset	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1

10.13.68.3 Fields

Field	Description
31-28 —	Reserved
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
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.

10.13.69 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).

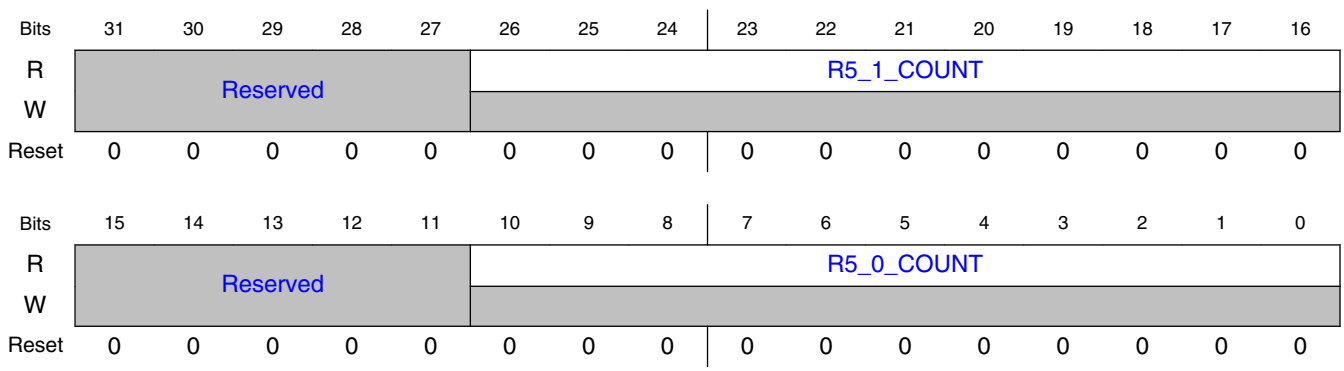
NOTE

This offset (0634h) is used as RTSCR5L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR5C readback register, as described here.

10.13.69.1 Offset

Register	Offset	Description
RTSCR5C	634h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.69.2 Diagram



10.13.69.3 Fields

Field	Description
31-27 —	Reserved
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
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.

10.13.70 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

This address (0634h) is used as RTSCR5L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR5C readback register.

10.13.70.1 Offset

Register	Offset	Description
RTSCR5L	634h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.70.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved					RUN5_RNG										
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved					RUN5_MAX										
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1

10.13.70.3 Fields

Field	Description
31-27 —	Reserved
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
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.

10.13.71 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).

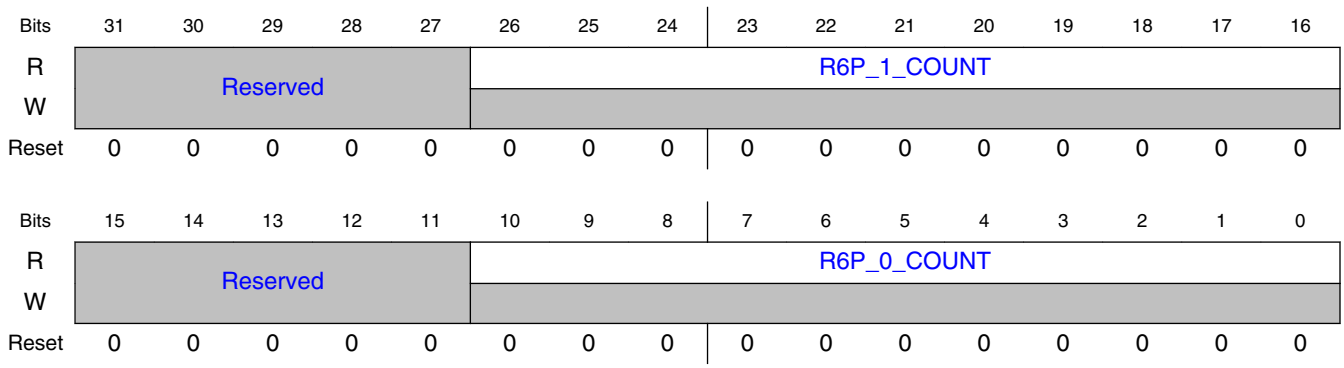
NOTE

This offset (0638h) is used as RTSCR6PL if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register, as described here.

10.13.71.1 Offset

Register	Offset	Description
RTSCR6PC	638h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.71.2 Diagram



10.13.71.3 Fields

Field	Description
31-27 —	Reserved
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
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.

10.13.72 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

This offset (0638h) is used as RTSCR6PL only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register.

10.13.72.1 Offset

Register	Offset	Description
RTSCR6PL	638h	Accessible at this address when RTMCTL[PRGM] = 1]

10.13.72.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved					RUN6P_RNG										
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved					RUN6P_MAX										
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1

10.13.72.3 Fields

Field	Description
31-27 —	Reserved
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
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.

10.13.73 RNG TRNG Status Register (RTSTATUS)

Various statistical tests are run as a normal part of the TRNG's entropy generation process. If the RNG TRNG Miscellaneous Control Register (RTMCTL) ERR field indicates an error, the least-significant 16 bits of the RTSTATUS register will indicate which test(s) have failed. 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 RTMCTL. If RTMCTL[ERR] indicates no error, then RTSTATUS register does not contain valid test status data.

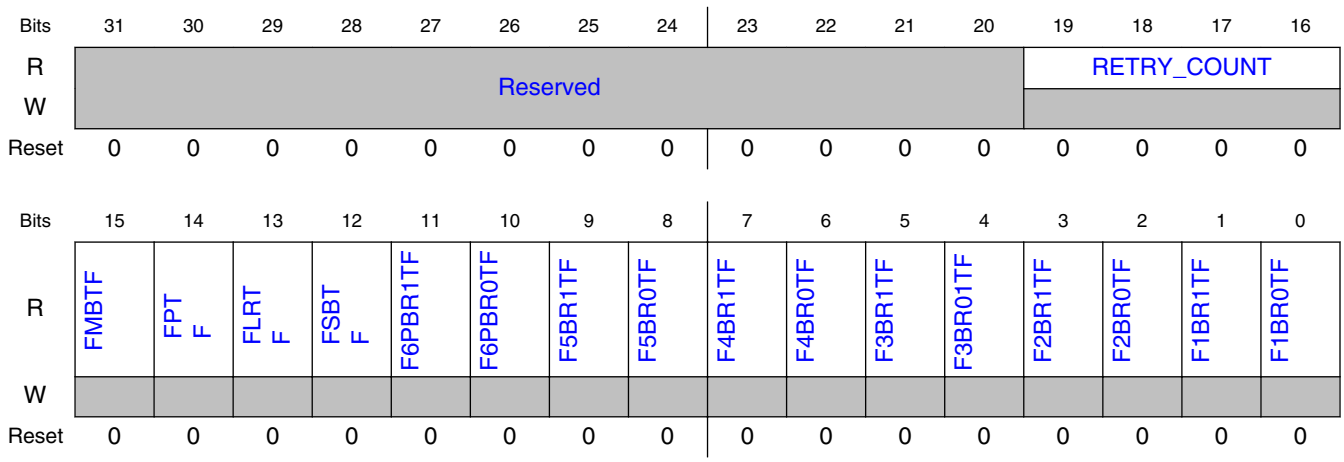
Note that there is a very small probability that a statistical test will fail even though the TRNG is operating properly. If this happens the TRNG will automatically retry the entire entropy generation process, including running all the statistical tests. The value in RETRY_COUNT is decremented each 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 RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)).

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 RNG TRNG Miscellaneous Control Register (RTMCTL)). If this register is read while the RNG is in Run Mode the value returned will be formatted as follows.

10.13.73.1 Offset

Register	Offset
RTSTATUS	63Ch

10.13.73.2 Diagram



10.13.73.3 Fields

Field	Description
31-20 —	Reserved
19-16 RETRY_COUNT	RETRY COUNT. This represents the current number of entropy generation retries left before a statistical text failure will cause the RNG to generate an error condition.
15 FMBTF	Mono Bit Test Fail. If MBTF=1, the Mono Bit Test has failed.
14 FPTF	Poker Test Fail. If PTF=1, the Poker Test has failed.
13 FLRTF	Long Run Test Fail. If LRTF=1, the Long Run Test has failed.
12 FSBTF	Sparse Bit Test Fail. If SBTF=1, the Sparse Bit Test has failed.
11 F6PBR1TF	6 Plus Bit Run, Sampling 1s, Test Fail. If 6PBR1TF=1, the 6 Plus Bit Run, Sampling 1s Test has failed.
10 F6PBR0TF	6 Plus Bit Run, Sampling 0s, Test Fail. If 6PBR0TF=1, the 6 Plus Bit Run, Sampling 0s Test has failed.
9 F5BR1TF	5-Bit Run, Sampling 1s, Test Fail. If 5BR1TF=1, the 5-Bit Run, Sampling 1s Test has failed.
8 F5BR0TF	5-Bit Run, Sampling 0s, Test Fail. If 5BR0TF=1, the 5-Bit Run, Sampling 0s Test has failed.
7	4-Bit Run, Sampling 1s, Test Fail. If 4BR1TF=1, the 4-Bit Run, Sampling 1s Test has failed.

Table continues on the next page...

Field	Description
F4BR1TF	
6 F4BR0TF	4-Bit Run, Sampling 0s, Test Fail. If 4BR0TF=1, the 4-Bit Run, Sampling 0s Test has failed.
5 F3BR1TF	3-Bit Run, Sampling 1s, Test Fail. If 3BR1TF=1, the 3-Bit Run, Sampling 1s Test has failed.
4 F3BR01TF	3-Bit Run, Sampling 0s, Test Fail. If 3BR0TF=1, the 3-Bit Run, Sampling 0s Test has failed.
3 F2BR1TF	2-Bit Run, Sampling 1s, Test Fail. If 2BR1TF=1, the 2-Bit Run, Sampling 1s Test has failed.
2 F2BR0TF	2-Bit Run, Sampling 0s, Test Fail. If 2BR0TF=1, the 2-Bit Run, Sampling 0s Test has failed.
1 F1BR1TF	1-Bit Run, Sampling 1s, Test Fail. If 1BR1TF=1, the 1-Bit Run, Sampling 1s Test has failed.
0 F1BR0TF	1-Bit Run, Sampling 0s, Test Fail. If 1BR0TF=1, the 1-Bit Run, Sampling 0s Test has failed.

10.13.74 RNG TRNG Entropy Read Register (RTENT0 - RTENT15)

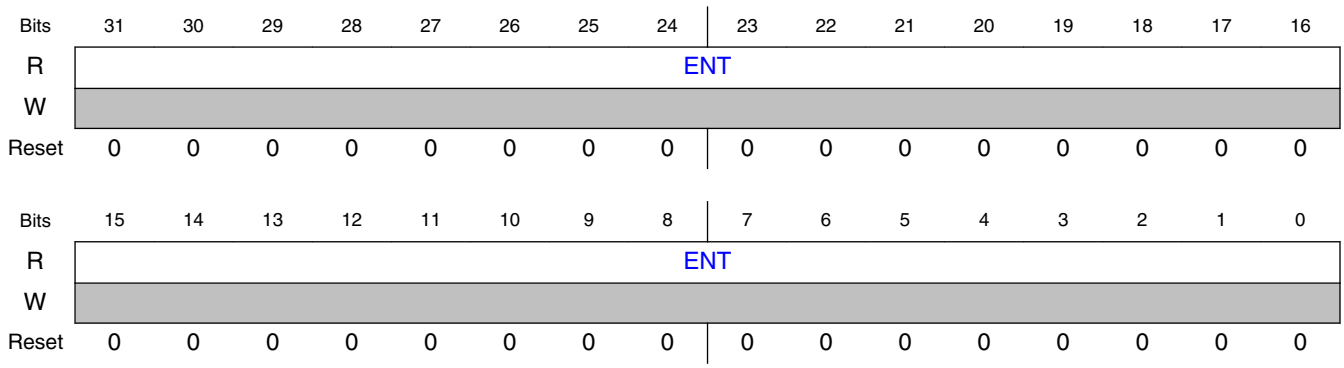
The RNG TRNG can be programmed to generate an entropy value that is readable via the SkyBlue bus. To do this, set the RTMCTL[TRNG_ACC] bit to 1. Once the entropy value has been generated, the RTMCTL[ENT_VAL] bit will be set to 1. At this point, RTENT0 through RTENT15 may be read to retrieve the 512-bit entropy value. Note that once RTENT15 is read, the entropy value will be cleared and a new value will begin generation, so it is important that RTENT15 be read last. Also note that the entropy value read from the RTENT0 - RTENT15 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.

10.13.74.1 Offset

For a = 0 to 15:

Register	Offset
RTENTa	640h + (a × 4h)

10.13.74.2 Diagram



10.13.74.3 Fields

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 the lowest offset, and the least significant bits are read from the highest offset. Note that reading the highest offset also clears the entire entropy value, and starts a new entropy generation.

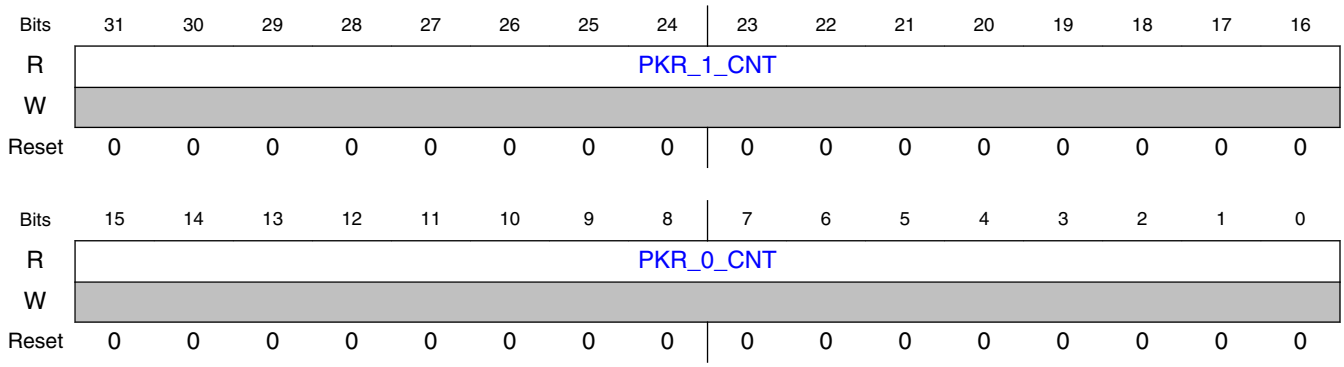
10.13.75 RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)

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.

10.13.75.1 Offset

Register	Offset
RTPKRCNT10	680h

10.13.75.2 Diagram



10.13.75.3 Fields

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.

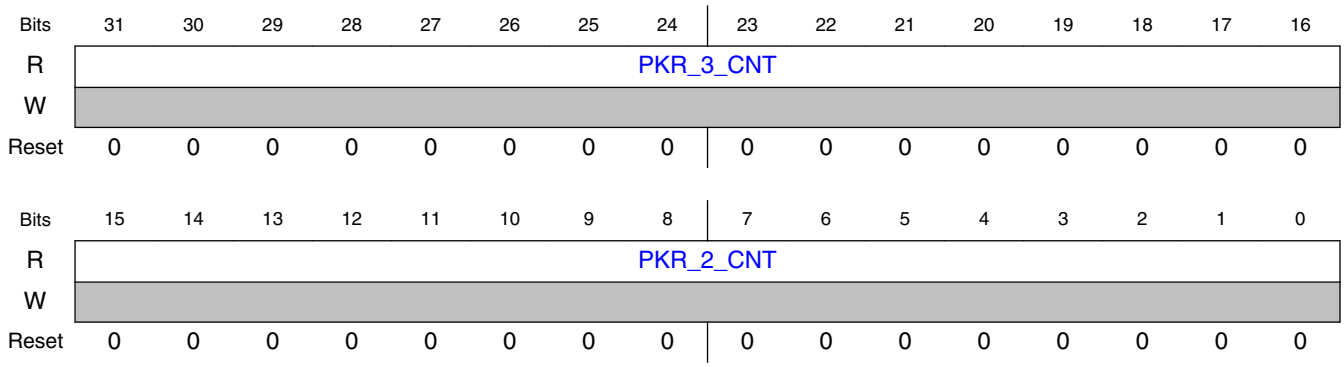
10.13.76 RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)

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.

10.13.76.1 Offset

Register	Offset
RTPKRCNT32	684h

10.13.76.2 Diagram



10.13.76.3 Fields

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.

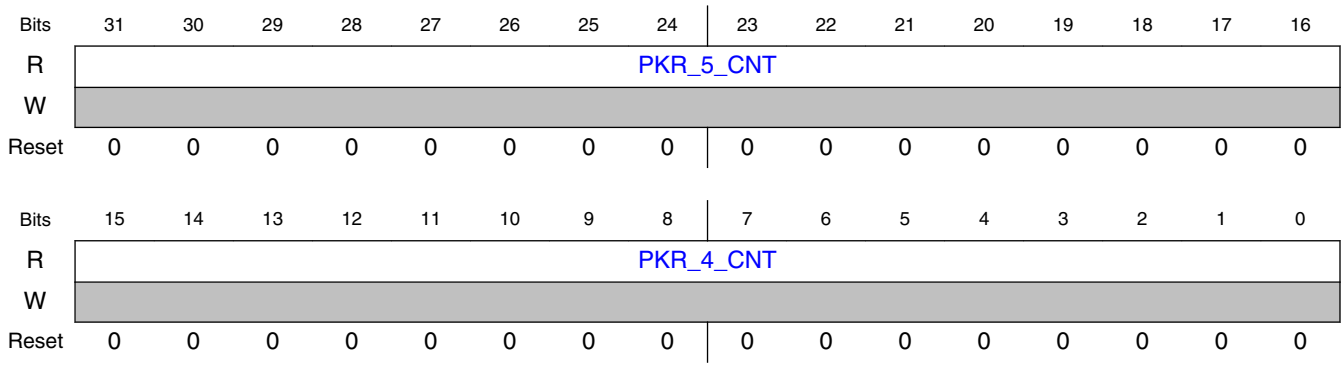
10.13.77 RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)

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.

10.13.77.1 Offset

Register	Offset
RTPKRCNT54	688h

10.13.77.2 Diagram



10.13.77.3 Fields

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.

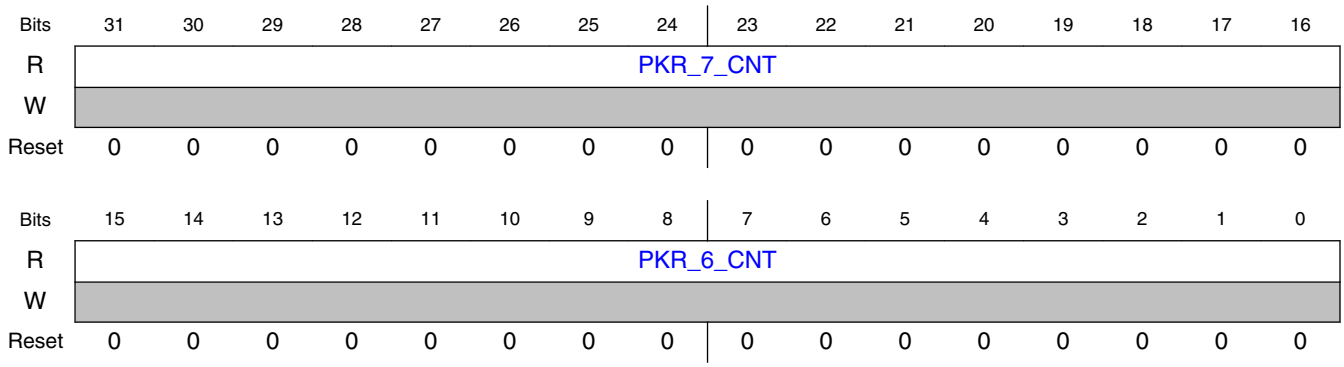
10.13.78 RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)

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.

10.13.78.1 Offset

Register	Offset
RTPKRCNT76	68Ch

10.13.78.2 Diagram



10.13.78.3 Fields

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.

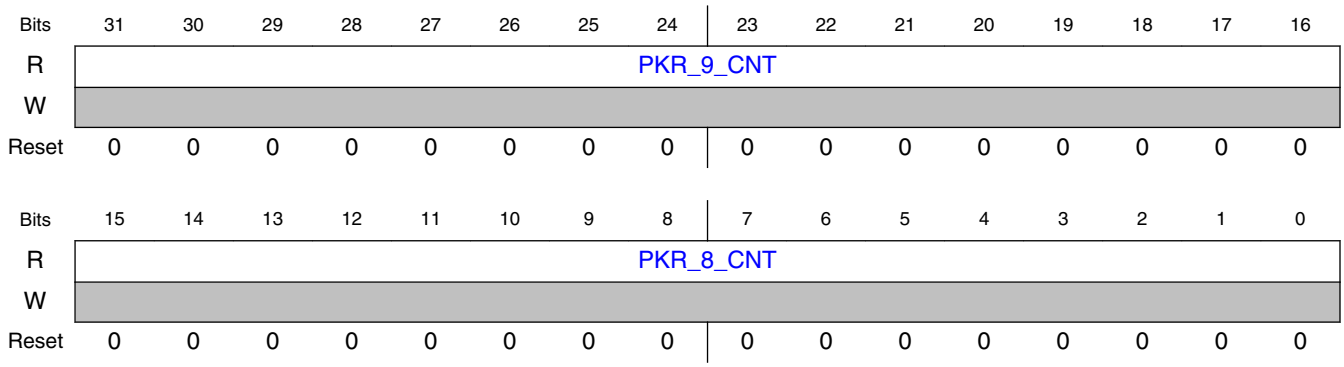
10.13.79 RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)

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.

10.13.79.1 Offset

Register	Offset
RTPKRCNT98	690h

10.13.79.2 Diagram



10.13.79.3 Fields

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.

10.13.80 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.

10.13.80.1 Offset

Register	Offset
RTPKRCNTBA	694h

10.13.80.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKR_B_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PKR_A_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.80.3 Fields

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.

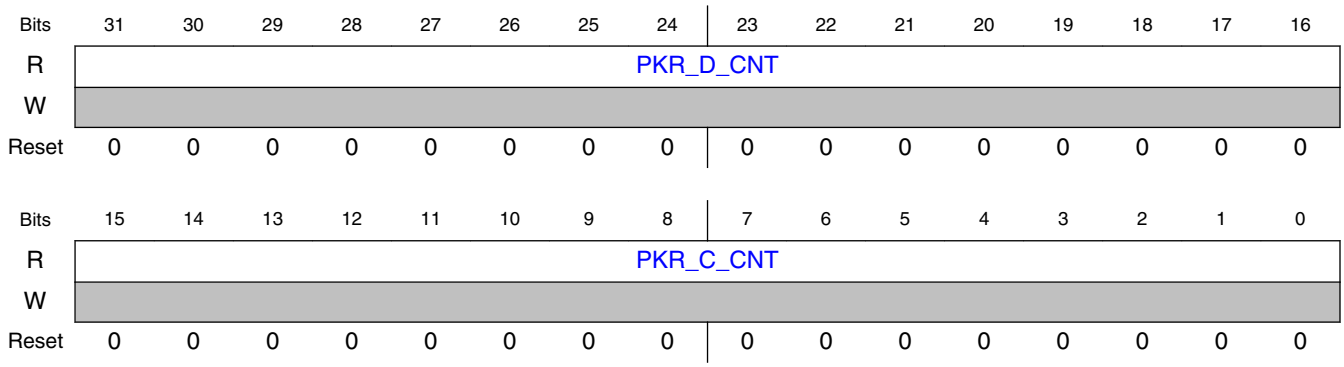
10.13.81 RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)

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.

10.13.81.1 Offset

Register	Offset
RTPKRCNTDC	698h

10.13.81.2 Diagram



10.13.81.3 Fields

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.

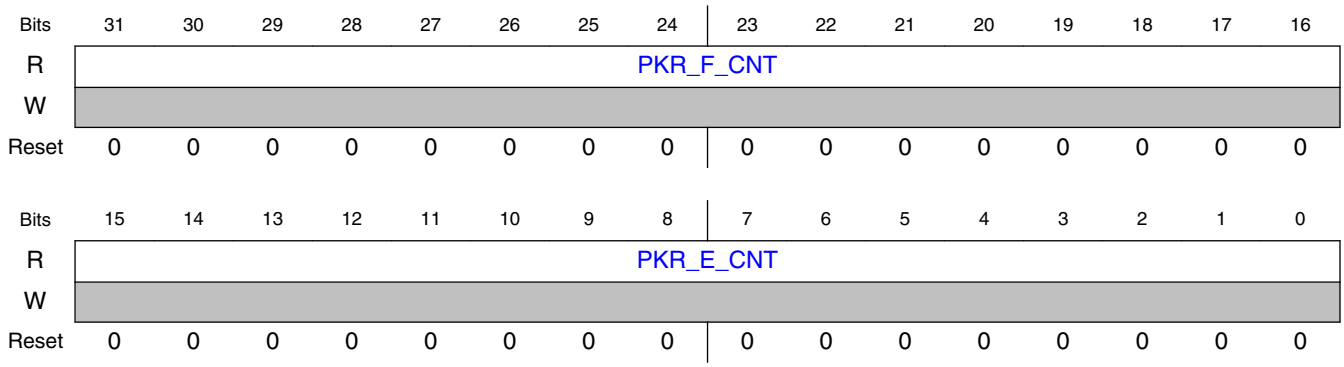
10.13.82 RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)

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 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.

10.13.82.1 Offset

Register	Offset
RTPKRCNTFE	69Ch

10.13.82.2 Diagram



10.13.82.3 Fields

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.

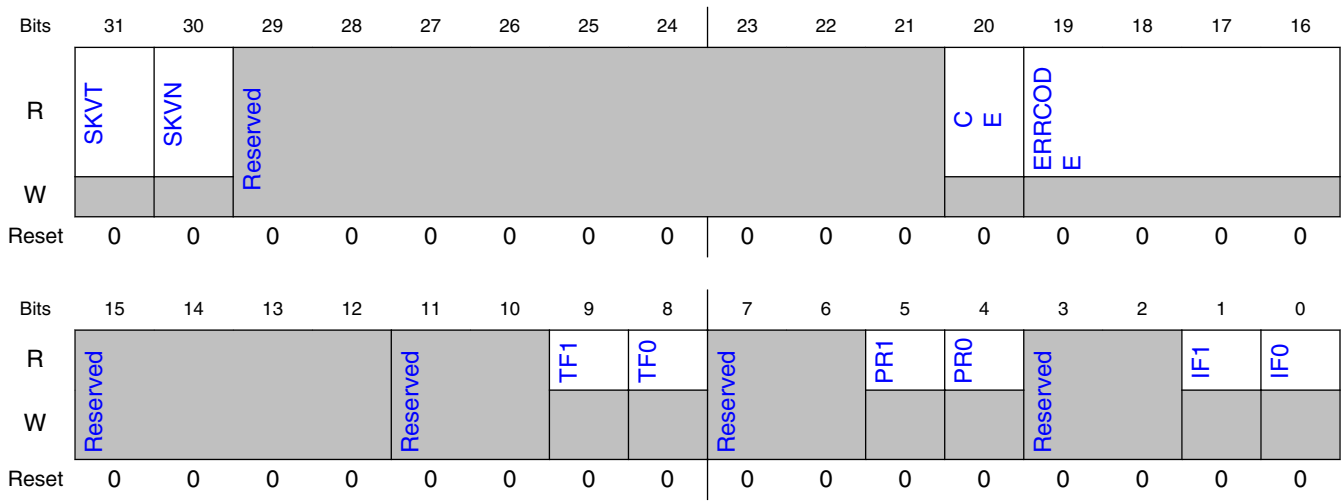
10.13.83 RNG DRNG Status Register (RDSTA)

The RNG DRNG Status Register shows the current status of the DRNG portion of the RNG.

10.13.83.1 Offset

Register	Offset	Description
RDSTA	6C0h	Accessible at this address when RTMCTL[PRGM] = 0]

10.13.83.2 Diagram



10.13.83.3 Fields

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
11-10 —	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.
7-6 —	Reserved

Table continues on the next page...

Field	Description
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.
3-2 —	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.

10.13.84 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 RNG operations).

10.13.84.1 Offset

Register	Offset
RDINT0	6D0h

10.13.84.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RESINT0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RESINT0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.84.3 Fields

Field	Description
31-0 RESINT0	RESINT0. This read-only register holds the Reseed Interval for State Handle 0.

10.13.85 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 RNG operations).

10.13.85.1 Offset

Register	Offset
RDINT1	6D4h

10.13.85.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RESINT1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RESINT1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.85.3 Fields

Field	Description
31-0 RESINT1	RESINT1. This read-only register holds the Reseed Interval for State Handle 1.

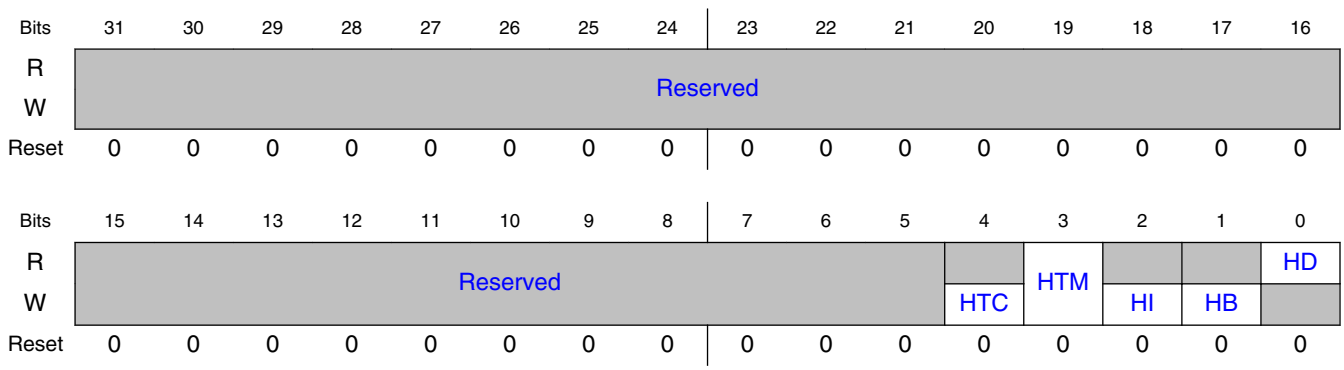
10.13.86 RNG DRNG Hash Control Register (RDHCNTL)

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.

10.13.86.1 Offset

Register	Offset
RDHCNTL	6E0h

10.13.86.2 Diagram



10.13.86.3 Fields

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 causing the Hashing Engine to begin hashing.
0 HD	Hashing Done. This bit asserts when the hashing engine is done.

10.13.87 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.

10.13.87.1 Offset

Register	Offset
RDHDIG	6E4h

10.13.87.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HASHMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HASHMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.87.3 Fields

Field	Description
31-0 HASHMD	HASHMD. Hashing Message Digest Register. This register needs to be read 8 times to retrieve the entire message digest.

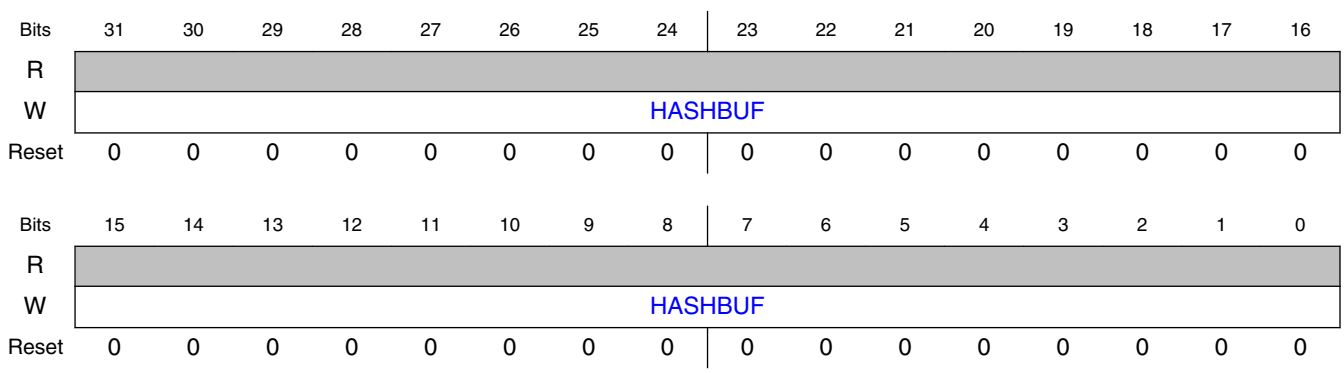
10.13.88 RNG DRNG Hash Buffer Register (RDHBUF)

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.

10.13.88.1 Offset

Register	Offset
RDHBUF	6E8h

10.13.88.2 Diagram



10.13.88.3 Fields

Field	Description
31-0 HASHBUF	HASHBUF. This write-only register provides access to the internal SHA-256 hashing engine's 64-byte buffer. This register must be written 16 times to fill the buffer.

10.13.89 Partition c SDID register (P0SDID_PG0 - P7SDID_JR2)

There is one Partition SDID (PSDID) register per Secure Memory partition. The PSDID register indicates the owner of the partition. When an unowned partition is claimed by writing into the partition's SMAP register, the SDID of the new owner is copied from an SDID register to the partition's PSDID register. If the SMAP register is written from an alias in a Job Ring page, the SDID value is copied from the Job Ring's JR(DID). If the SMAP register is written from register page 0, the SDID value is copied from the PAGE0_SDID register. Although the PSDID register can be read, the value in the PSDID register cannot be changed until the partition is released, at which time the PSDID register is cleared.

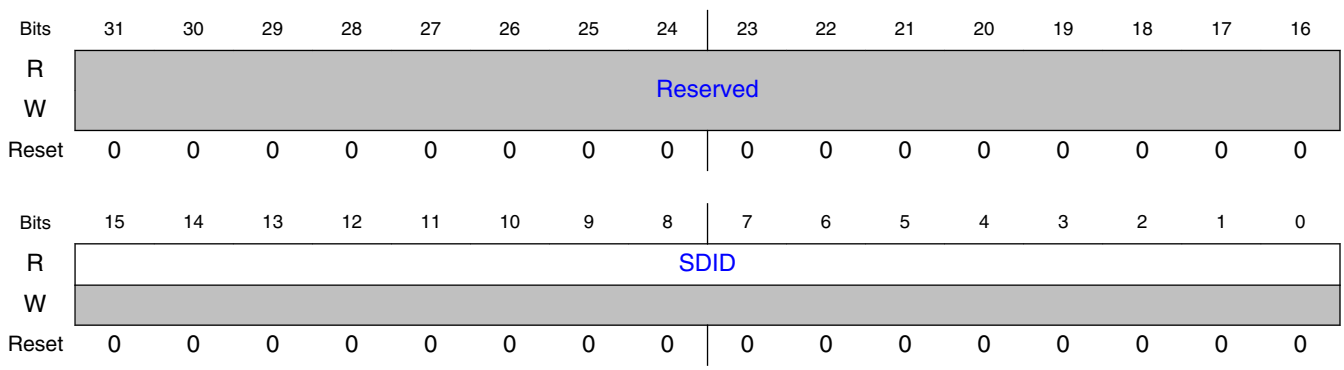
10.13.89.1 Offset

Register	Offset
P0SDID_PG0	A00h
P1SDID_PG0	A10h
P2SDID_PG0	A20h
P3SDID_PG0	A30h
P4SDID_PG0	A40h
P5SDID_PG0	A50h
P6SDID_PG0	A60h
P7SDID_PG0	A70h
P0SDID_JR0	1A00h
P1SDID_JR0	1A10h
P2SDID_JR0	1A20h
P3SDID_JR0	1A30h
P4SDID_JR0	1A40h
P5SDID_JR0	1A50h
P6SDID_JR0	1A60h
P7SDID_JR0	1A70h
P0SDID_JR1	2A00h
P1SDID_JR1	2A10h
P2SDID_JR1	2A20h
P3SDID_JR1	2A30h
P4SDID_JR1	2A40h
P5SDID_JR1	2A50h
P6SDID_JR1	2A60h
P7SDID_JR1	2A70h
P0SDID_JR2	3A00h
P1SDID_JR2	3A10h
P2SDID_JR2	3A20h

Table continues on the next page...

Register	Offset
P3SDID_JR2	3A30h
P4SDID_JR2	3A40h
P5SDID_JR2	3A50h
P6SDID_JR2	3A60h
P7SDID_JR2	3A70h

10.13.89.2 Diagram



10.13.89.3 Fields

Field	Description
31-16 —	Reserved
15-0 SDID	Security Domain Identifier. When a partition is claimed by a Job Ring owner or by the hypervisor or TrustZone SecureWorld by writing into the partition's Secure Memory Access Permissions Register, 0b, SDID_MS, 0b, PRIM_DID (if USE_OUT=0 and PRIM_TZ=0), or 100_0000_0000_1b, PRIM_DID (if USE_OUT=0 and PRIM_TZ=1), or SDID_MS, 0b, OUT_DID (if USE_OUT=1) is copied from the Job Ring's JRaDID register and written into the Partition SDID register for that partition. This indicates that the partition is owned by this particular security domain ID.

10.13.90 Secure Memory Access Permissions register (P0SMAPR_PG0 - P7SMAPR_JR2)

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 and one located in CAAM register page 0. Writing to the SMAPJR register sets the access permissions for the partition. If the partition is already owned, only the partition's owner[11](or the hypervisor or TrustZone SecureWorld) can write to the partition's SMAP register. If the partition is not currently owned, the partition can be claimed by writing to the partition's SMAP register. This records a new owner for the partition as well as setting the access permissions.

A Job Ring owner writes to the SMAP register using the address alias located in the Job Ring's register page. That is, the owner of Job Ring 0 would write to the SMAP address alias in register page 1, but the owner of Job Ring 1 would write to the SMAP address alias in register page 2. If the partition is unowned, at the time the SMAP register is written the partition's new owner (an SDID value) is recorded in the partition's PSDID register. Since a Job Ring is owned by a particular SDID value, when a Job Ring owner claims a partition the JR owner's SDID is copied from bits 14..0 of the JRDID register into the corresponding bits of the partition's PSDID register. Bit 4 (PRIM_TZ) of the JRDID register is copied into 15 of the partition's PSDIDR to indicate ownership by TrustZone SecureWorld or nonSecureWorld.

Similarly, the CAAM manager can claim an unowned partition by writing to the partition's SMAP alias in CAAM register page 0. In this case bits 14..0 of the PAGE0_SDID register are copied into the corresponding bits of the partition's PSDIDR, but bit 15 of the PSDIDR is set to 1 if the write to the SMAP register is from SecureWorld, or set to 0 if the write is from nonSecureWorld. This is enforced by comparing the DID value asserted during the write to the SMAPJR or SMAGJR register to the PRIM_DID field within the appropriate JRaDID 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 Access Group (SMAG2/1JR) registers. All bus masters and Job Descriptors acting on behalf of 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.

11. Bit 15 of the PSDIDR is ignored when determining read/write access to the partition's SMAP and SMAG registers or when reading the SMPO 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.

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 Register 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, Secure Memory Blob accesses, and key-reads. 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 DID 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. Note that a TrustZone Trusted Descriptor (i.e. a Trusted Descriptor created in a Job Ring owned by TrustZone SecureWorld) asserts the TrustZone security identifier, and this identifier is considered a memory of any access group. Therefore a TrustZone Trusted Descriptor can access any partition as long as at least one of the partition's Gx_TD bits is 1.

Because Trusted Descriptors can be 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, AFHA S-Box, or PKHA E Memory). 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 DID belongs), but key-read accesses are also permitted even if Gx_READ=0, as long as the DID 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 operating system or hypervisor or TrustZone SecureWorld could set the CPU's MMU to allow trusted software to read and write the partition, but prevent untrusted software from accessing the partition except via job descriptor key-reads. Note that the SMAPJR and SMAG2/1JR register permissions can also be set so that only particular bus masters (including CAAM itself) can access the partition. The table below indicates the various read and key-read permissions that can be defined.

Table 10-244. Read and Key-Read Permissions

	G1_READ=1	G1_READ=0
G2_READ=1	read or key-read permitted if DID is a member of G2 read or key-read permitted if DID is a member of G1	read or key-read permitted if DID is a member of G2 key-read permitted if DID is a member of G1
G2_READ=0	key-read permitted if DID is a member of G2 read or key-read permitted if DID is a member of G1	key-read permitted if DID is a member of G2 key-read permitted if DID 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 (see Register-based service interface).	

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 (if enabled by the appropriate Gx_TD setting) 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 SM APJR and SMAG2/1JR registers have the same values as the partition from which the Secure Memory Blob was exported.[12] 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.

The semantics of the access control bits can be summarized in the following Boolean equations:

Access_Permitted =

(If USE_OUT=0, or read access: PRIM_TZ,PRIM_DID ∈{ ACCESS_GROUP1 }) &

(If USE_OUT=1 and write access: 0,OUT_DID ∈ { ACCESS_GROUP1 }) &

(key_read |

(sm_blob_access & G1_SMBLOB) |

(caam_trusted_descriptor & G1_TD) |

(write & G1_WRITE) |

(read & G1_READ)))

(If USE_OUT=0, or read access: PRIM_TZ,PRIM_DID ∈ { ACCESS_GROUP2 }) &

(If USE_OUT=1 and write access: 0,OUT_DID ∈ { ACCESS_GROUP2 }) &

(key_read |

(sm_blob_access & G2_SMBLOB) |

(caam_trusted_descriptor & G2_TD) |

(write & G2_WRITE) |

(read & G2_READ)))

12. Observe that the partition owner's SDID appears in the Partition SDID register. This supports data privacy by preventing a Secure Memory Blob from being imported into a partition owned by a different Job Ring owner.

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 using the Critical Security Parameters bit[13] (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.

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 or has access to register page 0. But the changes to either the SMAPJR register or the SMAG2JR and SMAG1JR registers can be prevented by setting the SMAPJR_LCK" or SMAG1/2_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 SDID 000h, 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 DID 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.

13. The terms "Critical Security Parameter" and "Public Security Parameter" are defined in FIPS 140-3.

NOTE

At POR all SMAP registers reset to 000000FFh (which allows all access types) so that by default Secure Memory can be treated as ordinary RAM. However, due to CAAM's access control this may not be the value observed when the SMAP registers are read via the IP bus. If a partition's SMAP register is read by the owner of the partition using the SMAP register alias located in the owner's Job Ring page, the value read will be the SMAP register's actual content. If a nonowner reads the partition's SMAP register, or if the partition owner reads the partition's SMAP register using an address alias in a Job Ring register page that it doesn't own, the value 00000000h will be returned by the CAAM hardware. This is intended to conceal the partition's access control settings from everyone other than the partition owner. At POR partition 0 is allocated to SDID=0, and all DID registers are initialized so that they are associated with SDID=0 (i.e. SDID_MS,PRIM_TZ,PRIM_DID=0). So when the SMAP register for partition 0 is read from any CAAM Job Ring register page using DID value 0, the value 000000FFh will be returned. But if the SMAP register for partition 0 is read using a DID value other than 0, the value 00000000h will be returned. Since at POR all the other partitions are unowned, reading their SMAP registers will return 00000000h regardless of the DID used or the SMAP alias used.

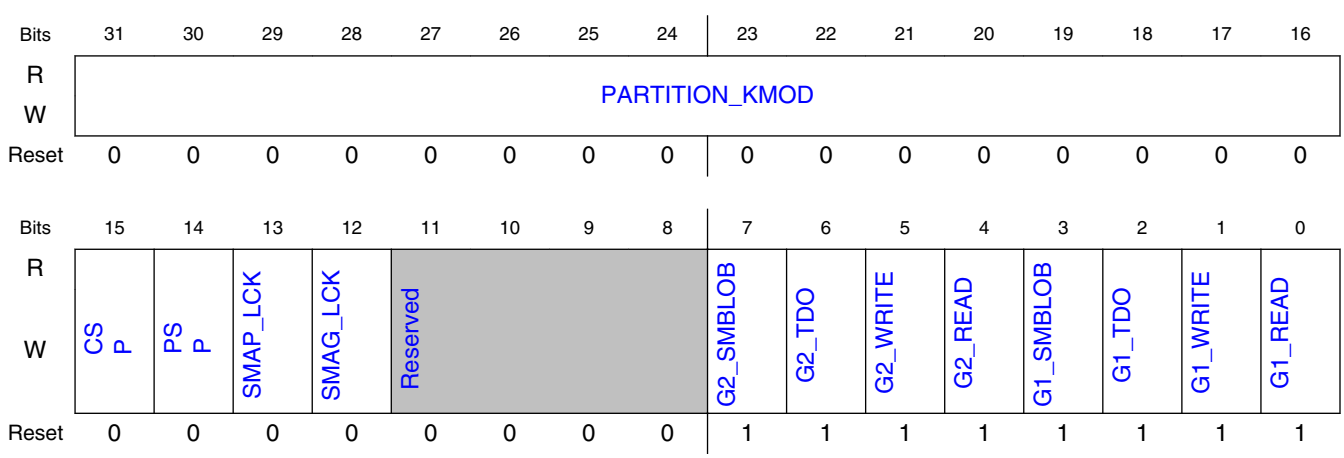
10.13.90.1 Offset

Register	Offset
P0SMAPR_PG0	A04h
P1SMAPR_PG0	A14h
P2SMAPR_PG0	A24h
P3SMAPR_PG0	A34h
P4SMAPR_PG0	A44h
P5SMAPR_PG0	A54h
P6SMAPR_PG0	A64h
P7SMAPR_PG0	A74h
P0SMAPR_JR0	1A04h
P1SMAPR_JR0	1A14h
P2SMAPR_JR0	1A24h
P3SMAPR_JR0	1A34h

Table continues on the next page...

Register	Offset
P4SMAPR_JR0	1A44h
P5SMAPR_JR0	1A54h
P6SMAPR_JR0	1A64h
P7SMAPR_JR0	1A74h
P0SMAPR_JR1	2A04h
P1SMAPR_JR1	2A14h
P2SMAPR_JR1	2A24h
P3SMAPR_JR1	2A34h
P4SMAPR_JR1	2A44h
P5SMAPR_JR1	2A54h
P6SMAPR_JR1	2A64h
P7SMAPR_JR1	2A74h
P0SMAPR_JR2	3A04h
P1SMAPR_JR2	3A14h
P2SMAPR_JR2	3A24h
P3SMAPR_JR2	3A34h
P4SMAPR_JR2	3A44h
P5SMAPR_JR2	3A54h
P6SMAPR_JR2	3A64h
P7SMAPR_JR2	3A74h

10.13.90.2 Diagram



10.13.90.3 Fields

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	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. 0 - The pages allocated to the partition will not be zeroized when they are de-allocated or the partition is released or a security alarm occurs. 1 - The pages allocated to the partition will be zeroized when they are individually de-allocated or the partition is released or a security alarm occurs.
14 PSP	Public Security Parameters. This field determines whether the partition and pages from the partition can be de-allocated. 0 - The partition and any of the pages allocated to the partition can be de-allocated. 1 - The partition cannot be de-allocated and the pages allocated to the partition cannot be de-allocated.
13 SMAP_LCK	SMAP LOCK bit. This field determines whether changes can be made to the partition's access control permissions. 0 - The SMAP register is unlocked. The partition owner can change any writable bits of the SMAP register. 1 - The SMAP register is locked. The SMAP_LCK, CSP and PSP bits and G1 and G2 permission bits of the SMAP 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.
12 SMAG_LCK	SMAG LOCK bit. This field determines whether changes can be made to the partition's access group definitions. 0 - The SMAG2JR register and SMAG1JR register are unlocked. The partition owner can change any writable bits of these registers. 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.
11-8 —	Reserved
7 G2_SMBLOB	Access Group 2 Secure Memory Blobs. Descriptors run from Job Rings whose SDID value matches the SDID in the partition's SMAGR2 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.) 0 - Exporting or importing Secure Memory Blobs is prohibited, unless done via a Trusted Descriptor and G2_TDO=1. 1 - Exporting or importing Secure Memory Blobs is allowed, regardless of the G2_READ and G2_WRITE settings.
6 G2_TDO	Access Group 2 Trusted Descriptor Override. Note that Trusted Descriptors can be executed only from a Job Ring whose SDID value matches the SDID value when the Trusted Descriptor was created. 0 - Trusted Descriptors have the same access privileges as Job Descriptors

Table continues on the next page...

Field	Description
	1 - Trusted Descriptors are allowed to override the other access permissions, i.e. they can export blobs from or import blobs to the partition and read from and write to the partition regardless of the G2_SMBLOB, G2_WRITE and G2_READ settings.
5 G2_WRITE	Access Group 2 Write. The bus masters whose DID bits are 1 in the partition's SMAG2JR register are 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 G2_SMBLOB=1 or if done by a Trusted Descriptor and G2_TDO=1). 1 - Writes are allowed (but importing a Secure Memory Blob is prohibited if G2_SMBLOB=0 and the descriptor is not a Trusted Descriptor or if G2_TDO=0).
4 G2_READ	Access Group 2 Read. The bus masters whose DID bits are 1 in the partition's SMAG2JR register are 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 (if G2_TDO=1) and key-reads are always allowed, and exporting Secure Memory Blobs is allowed if G2_SMBLOB=1 or if done by a Trusted Descriptor and G2_TDO=1). 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 or if G2_TDO=0).
3 G1_SMBLOB	Access Group 1 Secure Memory Blobs. Descriptors run from Job Rings whose SDID value matches the SDID in the partition's SMAGR1 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 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 and G1_TDO=1. 1 - Exporting or importing Secure Memory Blobs is allowed, regardless of the G1_READ and G1_WRITE settings.
2 G1_TDO	Access Group 1 Trusted Descriptor Override. Note that Trusted Descriptors can be executed only from a Job Ring whose SDID value matches the SDID value when the Trusted Descriptor was created. 0 - Trusted Descriptors have the same access privileges as Job Descriptors 1 - Trusted Descriptors are allowed to override the other access permissions, i.e. they can export blobs from or import blobs to the partition and read from and write to the partition regardless of the G1_SMBLOB, G1_WRITE and G1_READ settings.
1 G1_WRITE	Access Group 1 Write. The bus masters whose DID bits are 1 in the partition's SMAG1JR register are 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 and G1_TDO=1). 1 - Writes are allowed (but importing a Secure Memory Blob is prohibited if G1_SMBLOB=0 and the descriptor is not a Trusted Descriptor or if G1_TDO=0).
0 G1_READ	Access Group 1 Read. The bus masters whose DID bits are 1 in the partition's SMAG1JR register are 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 (if G1_TDO=1) and key-reads are always allowed, and exporting Secure Memory Blobs is allowed if G1_SMBLOB=1 or if done by a Trusted Descriptor and G1_TDO=1). 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 or if G1_TDO=0).

10.13.91 Secure Memory Access Group Registers (P0SMAG2_PG0 - P7SMAG1_JR2)

Each Secure Memory partition has a SMAG1 and a SMAG2 register. The SMAG1 and SMAG2 registers for each partition are aliased into the register page of each Job Ring. The SMAG1 and SMAG2 registers are accessible only by the partition's owner.[14] Unowned partitions can be claimed by writing to the partition's SMAP register.

For each allocated partition, the Secure Memory Access Group Register indicates which bus masters are allowed to access the partition. The SMAG1 and SMAG2 registers define two access groups, access group 1 and access group 2, corresponding to the two access permission sets defined in the SMAP register. The Secure Memory Access Group bit fields are defined so that each bit corresponds to a particular DID value. For example, the least-significant bit of each ACCESS_GROUP field represents DID 0h, and bit 15 represents DID Fh. Setting a bit to a 1 indicates that the DID identified by that particular value is a member of that access group. This encoding allows any combination of the DIDs to be included in either access group.

The nature of the DID associated with an access via the Secure Memory interface is determined by the particular hardware signals that are connected to CAAM's Secure Memory and register bus interfaces.

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 either the ns=!PRIM_TZ and the PRIM_DID value or ns=1 (nonSecureWorld) and the OUT_DID value specified within the appropriate JRxDID 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 TZ,DID value when performing DMA operations on behalf of jobs drawn from

14. Bit 15 of the PSDIDR is ignored when determining read/write access to the partition's SMAP and SMAG registers or when reading the SMPO register.

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 Secure Memory Access Permissions register (P0SMAPR_PG0 - P7SMAPR_JR2) 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 active in the PRIM_TZ and PRIM_DID fields of the job ring's JRaDID register). Attempts to write to the SMAPJR or SMAG2/1JR registers of a partition owned by another TZDID will be ignored, and read transactions from these registers will 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.

NOTE

At POR all SMAG registers reset to 0XFFFFFFFF so that by default Secure Memory can be treated as ordinary RAM. However, due to CAAM's access control this may not be the value observed when the SMAG registers are read via the IP bus. If a partition's SMAG registers are read by the owner of the partition using the SMAG register aliases located in the owner's Job Ring page, the value read will be the SMAG registers' actual content. If a nonowner reads the partition's SMAG registers, or if the partition owner reads the partition's SMAG registers using an address alias in a Job Ring register page that it doesn't own, the value 00000000h will be returned by the CAAM hardware. This is intended to conceal the partition's access control settings from everyone other than the partition owner. At POR partition 0 is allocated to Job Ring 0, and the JR0DID register is initialized so that the owner of Job Ring 0 is identified by DID= 0. So when the SMAG registers for partition 0 are read from CAAM register page 1 using DID value 0, the value 0XFFFFFFFF will be returned. If the SMAG registers for partition 0 are read from CAAM register page 1 using a DID value other than 0, the value 00000000h will be returned. Since at POR all the other partitions are unowned, reading their SMAG registers will return 00000000h regardless of the DID used or the SMAG alias used.

10.13.91.1 Offset

Register	Offset
P0SMAG2_PG0	A08h
P0SMAG1_PG0	A0Ch
P1SMAG2_PG0	A18h
P1SMAG1_PG0	A1Ch
P2SMAG2_PG0	A28h
P2SMAG1_PG0	A2Ch
P3SMAG2_PG0	A38h
P3SMAG1_PG0	A3Ch
P4SMAG2_PG0	A48h
P4SMAG1_PG0	A4Ch
P5SMAG2_PG0	A58h
P5SMAG1_PG0	A5Ch
P6SMAG2_PG0	A68h
P6SMAG1_PG0	A6Ch
P7SMAG2_PG0	A78h
P7SMAG1_PG0	A7Ch
P0SMAG2_JR0	1A08h
P0SMAG1_JR0	1A0Ch
P1SMAG2_JR0	1A18h
P1SMAG1_JR0	1A1Ch
P2SMAG2_JR0	1A28h
P2SMAG1_JR0	1A2Ch
P3SMAG2_JR0	1A38h
P3SMAG1_JR0	1A3Ch
P4SMAG2_JR0	1A48h
P4SMAG1_JR0	1A4Ch
P5SMAG2_JR0	1A58h
P5SMAG1_JR0	1A5Ch
P6SMAG2_JR0	1A68h
P6SMAG1_JR0	1A6Ch
P7SMAG2_JR0	1A78h
P7SMAG1_JR0	1A7Ch
P0SMAG2_JR1	2A08h
P0SMAG1_JR1	2A0Ch
P1SMAG2_JR1	2A18h
P1SMAG1_JR1	2A1Ch
P2SMAG2_JR1	2A28h
P2SMAG1_JR1	2A2Ch

Table continues on the next page...

Register	Offset
P3SMAG2_JR1	2A38h
P3SMAG1_JR1	2A3Ch
P4SMAG2_JR1	2A48h
P4SMAG1_JR1	2A4Ch
P5SMAG2_JR1	2A58h
P5SMAG1_JR1	2A5Ch
P6SMAG2_JR1	2A68h
P6SMAG1_JR1	2A6Ch
P7SMAG2_JR1	2A78h
P7SMAG1_JR1	2A7Ch
P0SMAG2_JR2	3A08h
P0SMAG1_JR2	3A0Ch
P1SMAG2_JR2	3A18h
P1SMAG1_JR2	3A1Ch
P2SMAG2_JR2	3A28h
P2SMAG1_JR2	3A2Ch
P3SMAG2_JR2	3A38h
P3SMAG1_JR2	3A3Ch
P4SMAG2_JR2	3A48h
P4SMAG1_JR2	3A4Ch
P5SMAG2_JR2	3A58h
P5SMAG1_JR2	3A5Ch
P6SMAG2_JR2	3A68h
P6SMAG1_JR2	3A6Ch
P7SMAG2_JR2	3A78h
P7SMAG1_JR2	3A7Ch

10.13.91.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Gx_ID31	Gx_ID30	Gx_ID29	Gx_ID28	Gx_ID27	Gx_ID26	Gx_ID25	Gx_ID24	Gx_ID23	Gx_ID22	Gx_ID21	Gx_ID20	Gx_ID19	Gx_ID18	Gx_ID17	Gx_ID16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Gx_ID15	Gx_ID14	Gx_ID13	Gx_ID12	Gx_ID11	Gx_ID10	Gx_ID09	Gx_ID08	Gx_ID07	Gx_ID06	Gx_ID05	Gx_ID04	Gx_ID03	Gx_ID02	Gx_ID01	Gx_ID00
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

10.13.91.3 Fields

Field	Description
31 Gx_ID31	Bit set to 1 indicates NonSecureWorld MID 15 is a member of Access Group x (1 or 2).
30 Gx_ID30	Bit set to 1 indicates NonSecureWorld MID 14 is a member of Access Group x (1 or 2).
29 Gx_ID29	Bit set to 1 indicates NonSecureWorld MID 13 is a member of Access Group x (1 or 2).
28 Gx_ID28	Bit set to 1 indicates NonSecureWorld MID 12 is a member of Access Group x (1 or 2).
27 Gx_ID27	Bit set to 1 indicates NonSecureWorld MID 11 is a member of Access Group x (1 or 2).
26 Gx_ID26	Bit set to 1 indicates NonSecureWorld MID 10 is a member of Access Group x (1 or 2).
25 Gx_ID25	Bit set to 1 indicates NonSecureWorld MID 09 is a member of Access Group x (1 or 2).
24 Gx_ID24	Bit set to 1 indicates NonSecureWorld MID 08 is a member of Access Group x (1 or 2).
23 Gx_ID23	Bit set to 1 indicates NonSecureWorld MID 07 is a member of Access Group x (1 or 2).
22 Gx_ID22	Bit set to 1 indicates NonSecureWorld MID 06 is a member of Access Group x (1 or 2).
21 Gx_ID21	Bit set to 1 indicates NonSecureWorld MID 05 is a member of Access Group x (1 or 2).
20 Gx_ID20	Bit set to 1 indicates NonSecureWorld MID 04 is a member of Access Group x (1 or 2).
19 Gx_ID19	Bit set to 1 indicates NonSecureWorld MID 03 is a member of Access Group x (1 or 2).
18 Gx_ID18	Bit set to 1 indicates NonSecureWorld MID 02 is a member of Access Group x (1 or 2).
17 Gx_ID17	Bit set to 1 indicates NonSecureWorld MID 01 is a member of Access Group x (1 or 2).
16 Gx_ID16	Bit set to 1 indicates NonSecureWorld MID 00 is a member of Access Group x (1 or 2).
15 Gx_ID15	Bit set to 1 indicates SecureWorld MID 15 is a member of Access Group x (1 or 2).
14	Bit set to 1 indicates SecureWorld MID 14 is a member of Access Group x (1 or 2).

Table continues on the next page...

Field	Description
Gx_ID14	
13 Gx_ID13	Bit set to 1 indicates SecureWorld MID 13 is a member of Access Group x (1 or 2).
12 Gx_ID12	Bit set to 1 indicates SecureWorld MID 12 is a member of Access Group x (1 or 2).
11 Gx_ID11	Bit set to 1 indicates SecureWorld MID 11 is a member of Access Group x (1 or 2).
10 Gx_ID10	Bit set to 1 indicates SecureWorld MID 10 is a member of Access Group x (1 or 2).
9 Gx_ID09	Bit set to 1 indicates SecureWorld MID 09 is a member of Access Group x (1 or 2).
8 Gx_ID08	Bit set to 1 indicates SecureWorld MID 08 is a member of Access Group x (1 or 2).
7 Gx_ID07	Bit set to 1 indicates SecureWorld MID 07 is a member of Access Group x (1 or 2).
6 Gx_ID06	Bit set to 1 indicates SecureWorld MID 06 is a member of Access Group x (1 or 2).
5 Gx_ID05	Bit set to 1 indicates SecureWorld MID 05 is a member of Access Group x (1 or 2).
4 Gx_ID04	Bit set to 1 indicates SecureWorld MID 04 is a member of Access Group x (1 or 2).
3 Gx_ID03	Bit set to 1 indicates SecureWorld MID 03 is a member of Access Group x (1 or 2).
2 Gx_ID02	Bit set to 1 indicates SecureWorld MID 02 is a member of Access Group x (1 or 2).
1 Gx_ID01	Bit set to 1 indicates SecureWorld MID 01 is a member of Access Group x.
0 Gx_ID00	Bit set to 1 indicates SecureWorld MID 00 is a member of Access Group x.

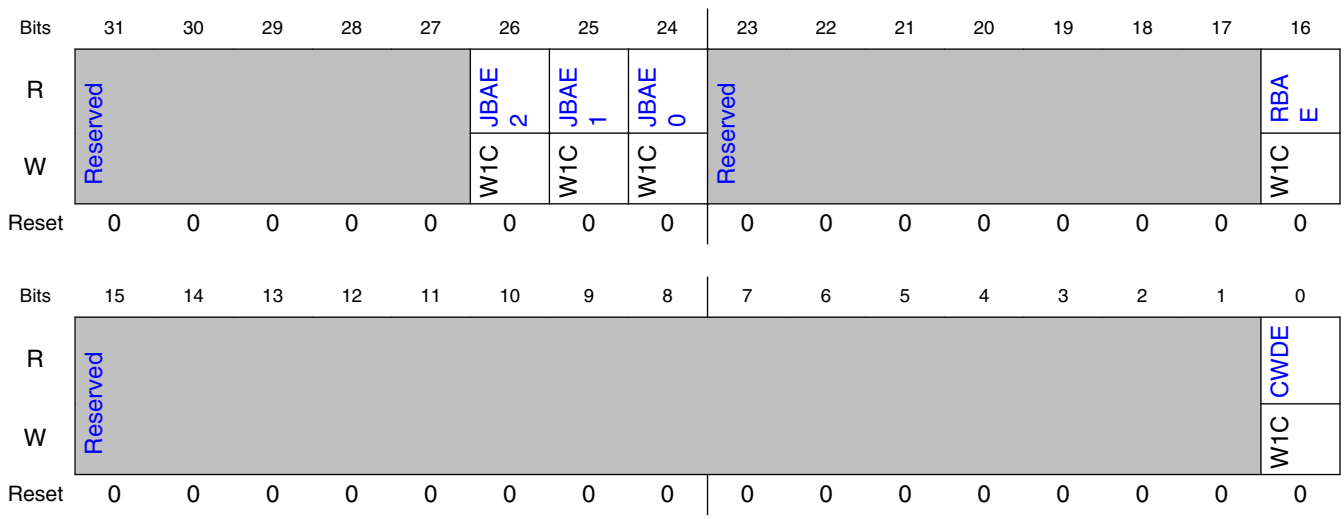
10.13.92 Recoverable Error Indication Status (REIS)

REIS indicates the assertion status of different CAAM recoverable error indication sources (1 bit per source). Software can clear a bit in REIS by writing a 1 to that bit.

10.13.92.1 Offset

Register	Offset
REIS	B00h

10.13.92.2 Diagram



10.13.92.3 Fields

Field	Description
31-27 —	Reserved
26 JBAE2	A job descriptor executed from Job Ring 2 caused a bus access error.
25 JBAE1	A job descriptor executed from Job Ring 1 caused a bus access error.
24 JBAE0	A job descriptor executed from Job Ring 0 caused a bus access error.
23-17 —	Reserved
16 RBAE	A bus transaction initiated by CAAM RTIC resulted in a bus access error.
15-1	Reserved

Table continues on the next page...

Field	Description
—	
0 CWDE	The CAAM watchdog timer expired.

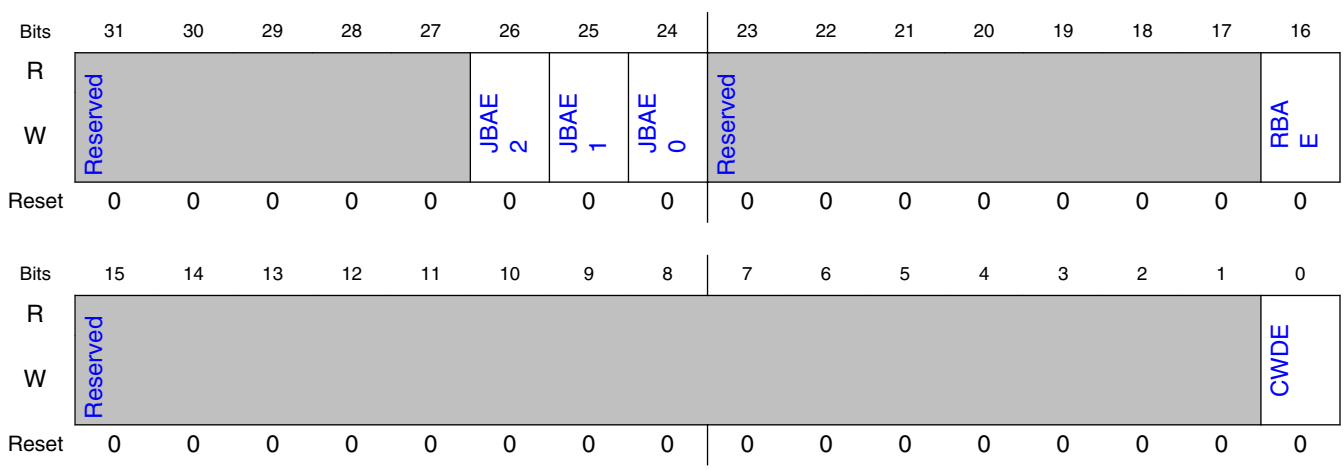
10.13.93 Recoverable Error Indication Halt (REIH)

Writing a 1 to an REIH bit indicates that CAAM should be halted if the associated recoverable error occurs.

10.13.93.1 Offset

Register	Offset
REIH	B0Ch

10.13.93.2 Diagram



10.13.93.3 Fields

Field	Description
31-27 —	Reserved

Table continues on the next page...

Field	Description
26 JBAE2	Halt CAAM if JR2-initiated job execution caused bus access error. 0 - Don't halt CAAM if JR2-initiated job execution caused bus access error. 1 - Halt CAAM if JR2-initiated job execution caused bus access error.
25 JBAE1	Halt CAAM if JR1-initiated job execution caused bus access error. 0 - Don't halt CAAM if JR1-initiated job execution caused bus access error. 1 - Halt CAAM if JR1-initiated job execution caused bus access error.
24 JBAE0	Halt CAAM if JR0-initiated job execution caused bus access error. 0 - Don't halt CAAM if JR0-initiated job execution caused bus access error. 1 - Halt CAAM if JR0-initiated job execution caused bus access error.
23-17 —	Reserved
16 RBAE	Halt CAAM if RTIC-initiated job execution caused bus access error. 0 - Don't halt CAAM if RTIC-initiated job execution caused bus access error. 1 - Halt CAAM if RTIC-initiated job execution caused bus access error.
15-1 —	Reserved
0 CWDE	Halt CAAM if CAAM watchdog timer expires. 0 - Don't halt CAAM if CAAM watchdog expired. 1 - Halt CAAM if CAAM watchdog expired..

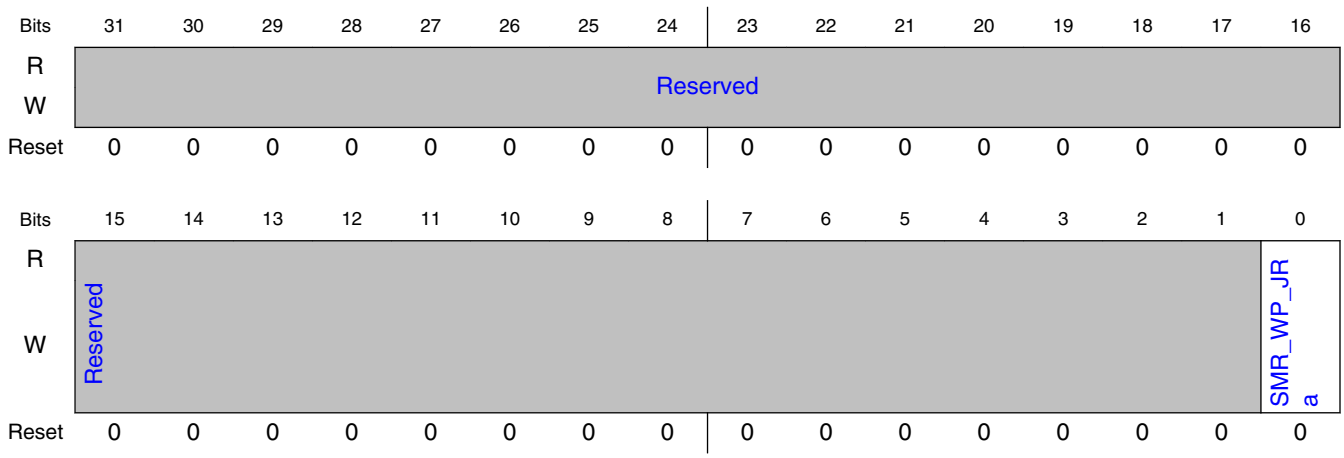
10.13.94 Secure Memory Write Protect Job Ring Register (SMWP JR0R - SMWPJR2R)

This register allows the CAAM manager to prevent changes to the configuration of Secure Memory.

10.13.94.1 Offset

Register	Offset
SMWPJR0R	BD0h
SMWPJR1R	BD4h
SMWPJR2R	BD8h

10.13.94.2 Diagram



10.13.94.3 Fields

Field	Description
31-1 —	Reserved
0 SMR_WP_JRa	Secure Memory Registers Write Protect. When SMR_WP_JRa = 1, writing into the Secure Memory Registers (SMAPR, SMAG1R, SMAG2R, SMCR) in the Job Ring a page is prohibited.

10.13.95 Secure Memory Command Register (SMCR_PG0 - SMCR_JR2)

The Secure Memory contains Partitions numbered 0 to 7, any of which can contain zero or more fixed-size pages. A functional description of the Secure Memory can be found in Secure memory.

At power up all Secure Memory pages are allocated to Partition 0, and ownership of Partition 0 is assigned to SDID 0. Any software entity that owns a Job Ring can claim an available partition by writing to the partition's SMAP register (see SectionSecure Memory Access Permissions register (P0SMAPR_PG0 - P7SMAPR_JR2)). 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 a can write to SMC_JRa. 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 SMC_JR register will be accepted only for partitions owned by the current Job Ring owner. Note that there is another Secure Memory Command Register located in CAAM register page 0 that can be used by the hypervisor to issue commands for any partition owned by any SDID within TrustZone nonSecureWorld, or can be used by TrustZone SecureWorld to issue commands for any partition owned by TrustZone nonSecureWorld or owned by any SDID within TrustZone nonSecureWorld.

Allocating a page associates that page with a partition. Access to that page is now controlled by the associated partition's SMAP and SMAG2/1 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.

10.13.95.1 Offset

Register	Offset	Description
SMCR_PG0	BE4h	(used by the hypervisor and TrustZone SecureWorld)
SMCR_JR0	1BE4h	(used by JR0)
SMCR_JR1	2BE4h	(used by JR1)
SMCR_JR2	3BE4h	(used by JR2)

10.13.95.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	PAGE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								Reserved							
W					PRTN								CMD			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.95.3 Fields

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 's current SDID does not own the partition, the command is ignored and an error will be shown in the Secure Memory Command Status Register. If the command is issued via register page 0 (i.e. offset 0BE4h), the command will be acted upon regardless of which SDID owns the partition. 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.
7-4 —	Reserved
3-0 CMD	Command: 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[1] by the requester the page will not be allocated and instead a partition ownership error will be returned. 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[1] by the requester the page will not be de-allocated and instead a partition ownership error will be returned.

Field	Description
	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[1] by the requester 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. 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. All other values: reserved Each allocated partition is owned by some Security Domain, which is indicated by the partition's PSDID register. Each Job Ring is owned by some Security Domain, which is indicated by the Job Ring's JRaDID register's SDID value. If Job Ring x's JRaDID[SDID] field matches partition y's PSDID register, then the owner can issue commands for partition y via the Secure Memory Command register located in Job Ring x's register page. Note that it is possible that the owner of a partition does not currently own a Job Ring, and so cannot issue commands for the partition. The hypervisor could park a Job Ring and allocate the Job Ring to the Security Domain when a command needs to be issued for that partition. But the hypervisor and TrustZone SecureWorld can also issue commands for any partition via the SMCR in register page 0. For commands issued via the page 0 SMCR there is no comparison of SDID values, so the hypervisor can issue commands for any non-SecureWorld partition and SecureWorld can issue commands for any partition whether or not it is owned by SecureWorld.Note that TrustZone SecureWorld is considered to belong to all Security Domains, so SecureWorld can also issue commands for any partition via any Job Ring's SMCR regardless of the JRaDID[SDID] setting. The hypervisor cannot issue commands for SecureWorld partitions because the write to SMCR must have ns=0 if a partition is owned by TrustZone SecureWorld, or a partition ownership error will be generated and the command will be ignored.

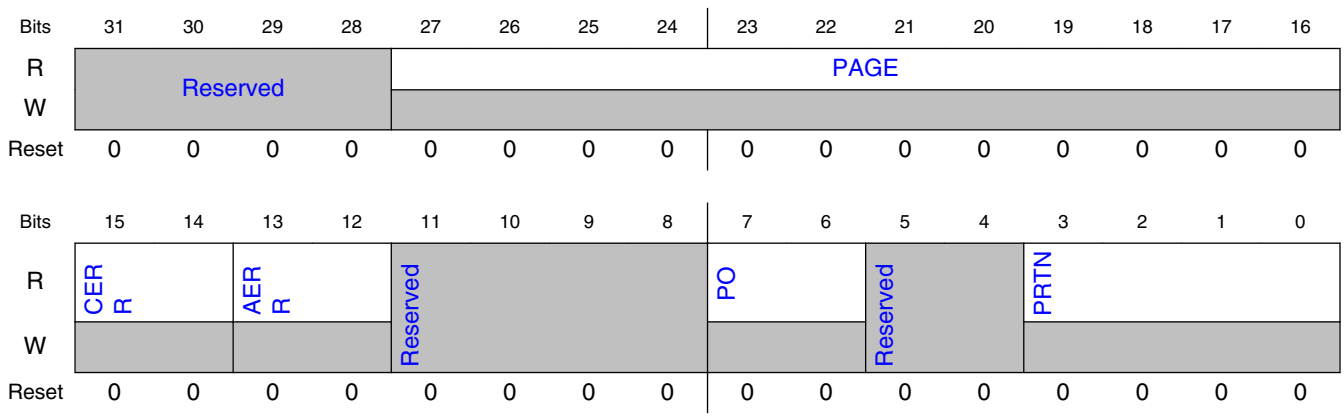
10.13.96 Secure Memory Command Status Register (SMCSR_PG 0 - SMCSR_JR2)

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 a can read from SMCS_JRa. 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.

10.13.96.1 Offset

Register	Offset	Description
SMCSR_PG0	BECh	(used by the hypervisor and TrustZone SecureWorld)
SMCSR_JR0	1BECh	(used by JR0)
SMCSR_JR1	2BECh	(used by JR1)
SMCSR_JR2	3BECh	(used by JR2)

10.13.96.2 Diagram



10.13.96.3 Fields

Field	Description
31-28 —	Reserved
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 SMC_JR register yielded an error, this field shows the error code. 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 continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>13-12
AERR</td><td>Allocation Error. If the command issued via the SMC_JR 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>
<tr><th>Value</th><th colspan="2">Description</th></tr>
<tr><th></th><th>After an Allocate Page Command</th><th>After a De-Allocate Page or a De-Allocate Partition Command</th></tr>
<tr><td>00</td><td colspan="2">No Error.</td></tr>
<tr><td>01</td><td>Reserved.</td><td>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>10</td><td>Page availability error. The page is either already allocated to a partition, is non-existent, or is not yet initialized.</td><td>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>11</td><td>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>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>
</table></td></tr>
<tr><td>11-8
—</td><td>Reserved</td></tr>
<tr><td>7-6
PO</td><td>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 SMC_JR register.
00 - Available; Unowned: 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.</td></tr>
<tr><td>5-4
—</td><td>Reserved</td></tr>
<tr><td>3-0
PRTN</td><td>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 SMC_JR register command, or if the PO field is 00 or 01 following a Page Inquiry Command, the value in this field is invalid.</td></tr>
</table>

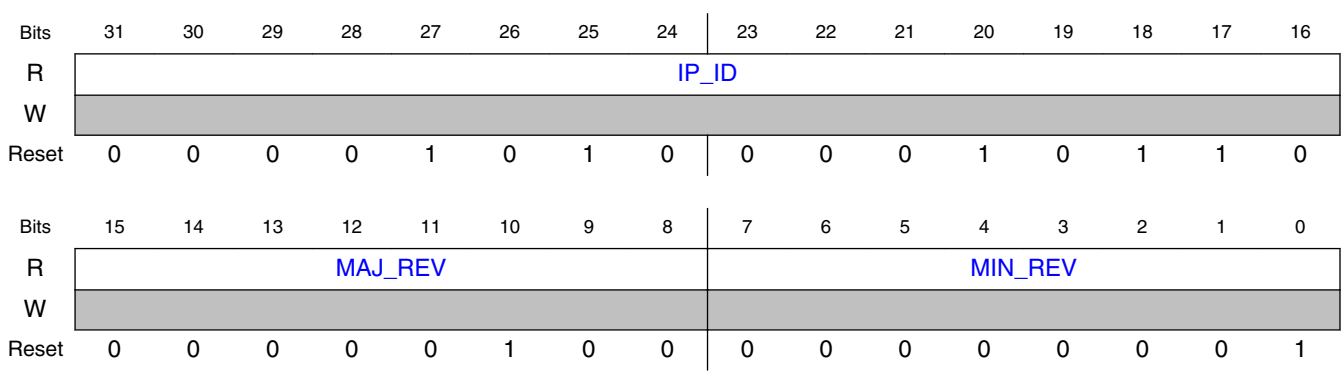
10.13.97 CAAM Version ID Register, most-significant half (CAAM VID_MS)

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 holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4KB address spaces. The register and its fields are described in the figure and table below.

10.13.97.1 Offset

Register	Offset
CAAMVID_MS	BF8h (alias)
CAAMVID_MS	FF8h (alias)
CAAMVID_MS	1FF8h (alias)
CAAMVID_MS	2FF8h (alias)
CAAMVID_MS	3FF8h (alias)
CAAMVID_MS	6FF8h (alias)
CAAMVID_MS	8FF8h (alias)

10.13.97.2 Diagram



10.13.97.3 Fields

Field	Description
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.

10.13.98 CAAM Version ID Register, least-significant half (CAAM VID_LS)

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 holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces. The register and its fields are described in the figure and table below.

10.13.98.1 Offset

Register	Offset
CAAMVID_LS	BFCh (alias)
CAAMVID_LS	FFCh (alias)
CAAMVID_LS	1FFCh (alias)
CAAMVID_LS	2FFCh (alias)
CAAMVID_LS	3FFCh (alias)
CAAMVID_LS	6FFCh (alias)
CAAMVID_LS	8FFCh (alias)

10.13.98.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	COMPILE_OPT								INTG_OPT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ECO_REV								CONFIG_OPT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.98.3 Fields

Field	Description
31-24 COMPILE_OPT	Compile options for CAAM.
23-16 INTG_OPT	Integration options for CAAM.
15-8 ECO_REV	ECO revision for CAAM.
7-0 CONFIG_OPT	Configuration options for CAAM.

10.13.99 Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR)

The HTa_JD_ADDR register holds the address of a Job 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 Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Job Descriptor Address to the HTa_JD_ADDR.

10.13.99.1 Offset

Register	Offset	Description
HT0_JD_ADDR	C00h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.99.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.99.3 Fields

Field	Description
63-32 —	Reserved
31-0 JD_ADDR	Job Descriptor Address.

10.13.100 Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)

The HTa_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.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Shared Descriptor Address to the HTa_SD_ADDR.

10.13.100.1 Offset

Register	Offset	Description
HT0_SD_ADDR	C08h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.100.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.100.3 Fields

Field	Description
63-32 —	Reserved
31-0 SD_ADDR	Shared Descriptor Address.

10.13.101 Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS)

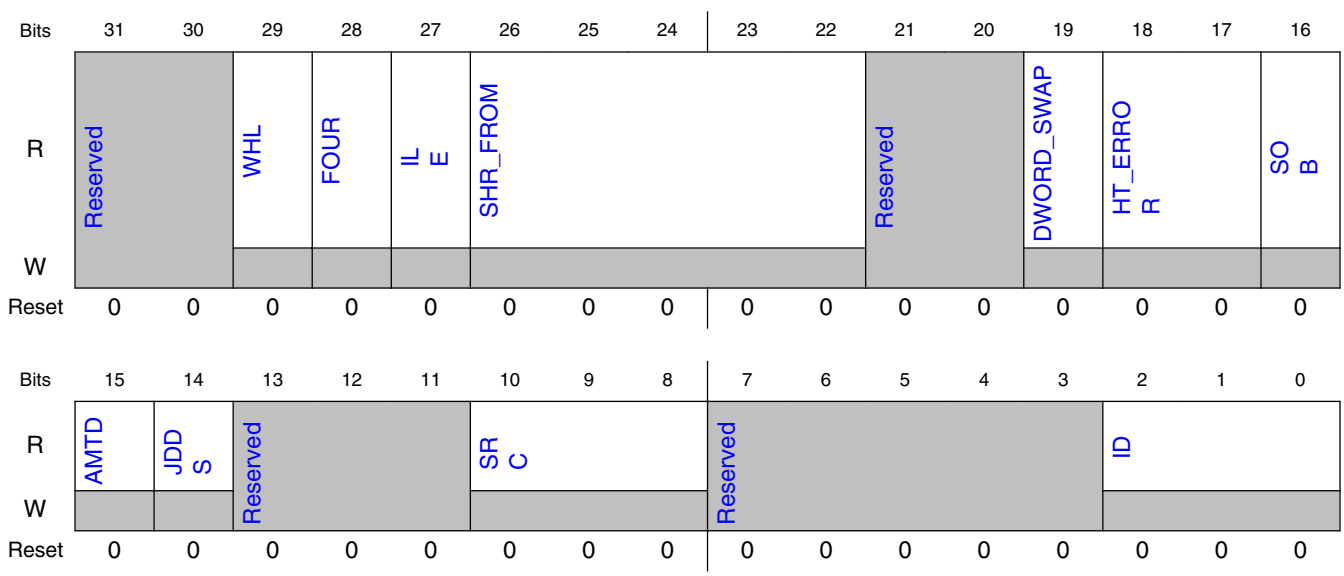
The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Job Queue control data of the HTa_JQ_CTRL_MS.

10.13.101.1 Offset

Register	Offset	Description
HT0_JQ_CTRL_MS	C10h	Note that the addresses of the two halves of this register are unaffected by the endianness configuration.

10.13.101.2 Diagram



10.13.101.3 Fields

Field	Description
31-30	Reserved

Table continues on the next page...

Field	Description
—	
29 WHL	Whole Descriptor. In versions of CAAM that implement prefetching, the WHL field is interpreted in combination with the SOB field. In versions that don't implement prefetching, WHL=1 indicates that HT is passing the full job descriptor to DECO and therefore DECO does not need to fetch any additional Job Descriptor words from external memory.
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. Byte-swapping is controlled when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: • Input Data FIFO • Output Data FIFO • Auxiliary Data FIFO • Class 1 Context register • Class 2 Context register • Class 1 Key register • Class 2 Key register 0 - No byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer. 1 - Byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer.
26-22 SHR_FROM	Share From. This is the DECO block from which the DECO block that runs this job 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).
21-20 —	Reserved
19 DWORD_SWAP	Double Word Swap. 0 - DWords are in the order most-significant word, least-significant word. 1 - DWords are in the order least-significant word, most-significant word.
18-17 HT_ERROR	Holding Tank Error. (This field is implemented only in versions of CAAM that support prefetching.) 00 - No error 01 - Job Descriptor or Shared Descriptor length error 10 - AXI_error while reading a Job Ring Shared Descriptor or the remainder of a Job Ring Job Descriptor 11 - reserved
16 SOB	Shared or Burst. (This field is implemented only in versions of CAAM that support prefetching.) The SOB field is interpreted along with the WHL field as follows: SOB=0 WHL=0 - No prefetch, not whole descriptor SOB=0 WHL=1 - Got whole Job Descriptor, no Shared Descriptor or input frame data SOB=1 WHL=0 - Got Shared Descriptor, no input frame data SOB=1 WHL=1 - Got whole Job Descriptor and input frame data
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 JRaDID Register.
14 JDDS	Job Descriptor DID Select. Determines whether the SEQ DID or the Non-SEQ DID is asserted when reading the Job Descriptor from memory. 0 - Non-SEQ DID

Table continues on the next page...

Field	Description
	1 - SEQ DID
13-11 —	Reserved
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS and endian configuration bits) the DMA should use for bus 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). 000 - Job Ring 0 001 - Job Ring 1 010 - Job Ring 2 011 - Reserved 100 - RTIC 101 - Reserved 110 - Reserved 111 - Reserved
7-3 —	Reserved
2-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.

10.13.102 Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)

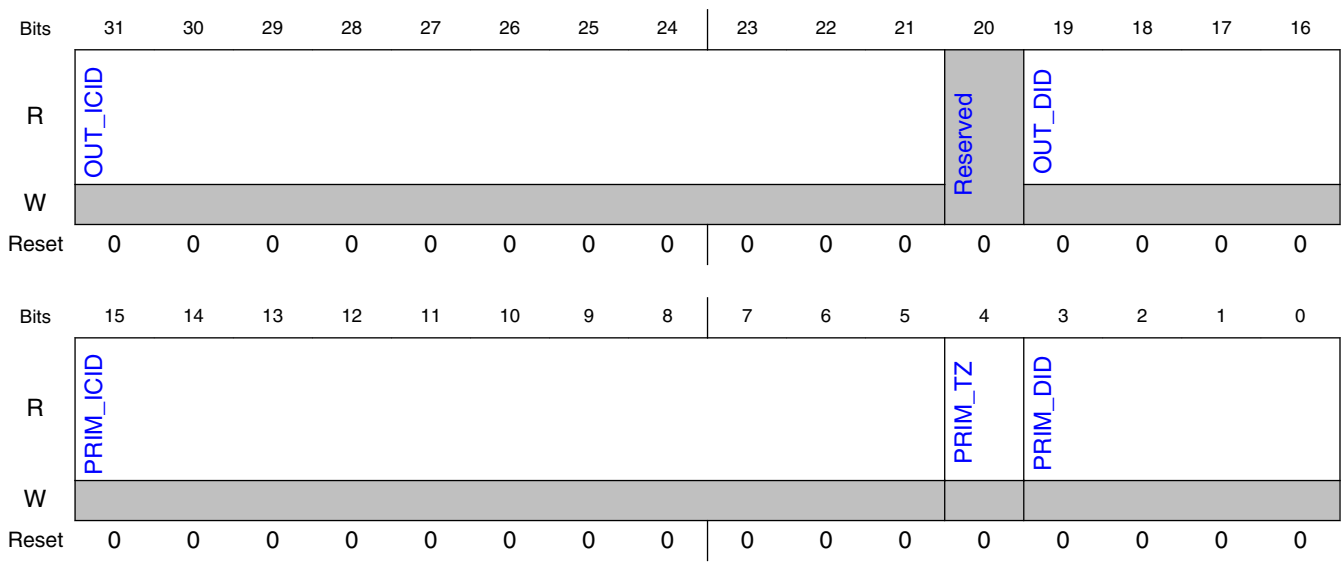
The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Job Queue control data to the HTa_JQ_CTRL_LS.

10.13.102.1 Offset

Register	Offset	Description
HT0_JQ_CTRL_LS	C14h	Note that the addresses of the two halves of this register are unaffected by the endianness configuration.

10.13.102.2 Diagram



10.13.102.3 Fields

Field	Description
31-21 OUT_ICID	Output ICID. When JRaDID[USE_OUT]=0, the value in this field is unused. When JRaDID[USE_OUT]=1, this ICID value is asserted for external memory data writes, but not for reads or for job completion status writes or descriptor write-backs.
20 —	Reserved
19-16 OUT_DID	Output DID. When JRaDID[USE_OUT]=0, the value in this field is unused. When JRaDID[USE_OUT]=1, this DID value is asserted for all external memory data writes, but not for reads or for job completion status writes or descriptor write-backs.
15-5 PRIM_ICID	Primary ICID. When JRaDID[USE_OUT]=0, the value in this field indicates the ICID asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this ICID value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs.

Table continues on the next page...

Field	Description
4 PRIM_TZ	Primary TZ. When JRaDID[USE_OUT]=0, the value in this field indicates the TrustZone World value (PRIM_TZ=1 means SecureWorld) asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this TZ value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs. 0 - TrustZone NonSecureWorld 1 - TrustZone SecureWorld
3-0 PRIM_DID	Primary DID. When JRaDID[USE_OUT]=0, the value in this field indicates the DID value asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this DID value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs.

10.13.103 Holding Tank Status (HT0_STATUS)

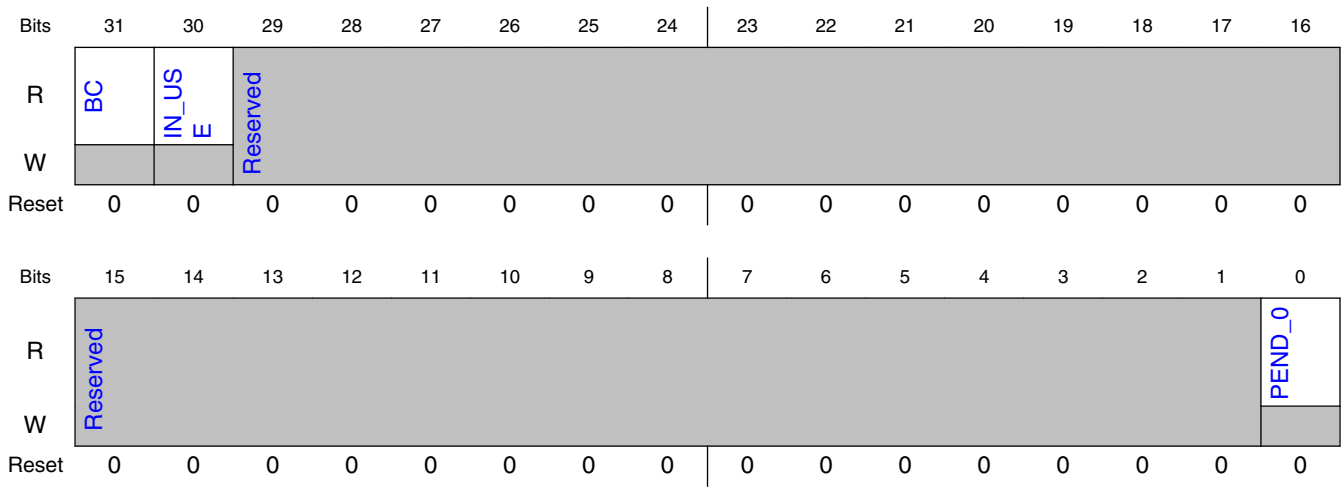
The HT0_STATUS register holds the status information for a Job Descriptor "holding tank". The register is intended to be used when debugging descriptor execution.

The HT_SEL field in the Job Queue Debug Select Register (JQ_DEBUG_SEL) controls which holding tank supplies the status information to the HT0_STATUS.

10.13.103.1 Offset

Register	Offset
HT0_STATUS	C1Ch

10.13.103.2 Diagram



10.13.103.3 Fields

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.
29-1 —	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.

10.13.104 Job Queue Debug Select Register (JQ_DEBUG_SEL)

The Job Queue Debug Select register is used to select which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers). The Job Queue Debug

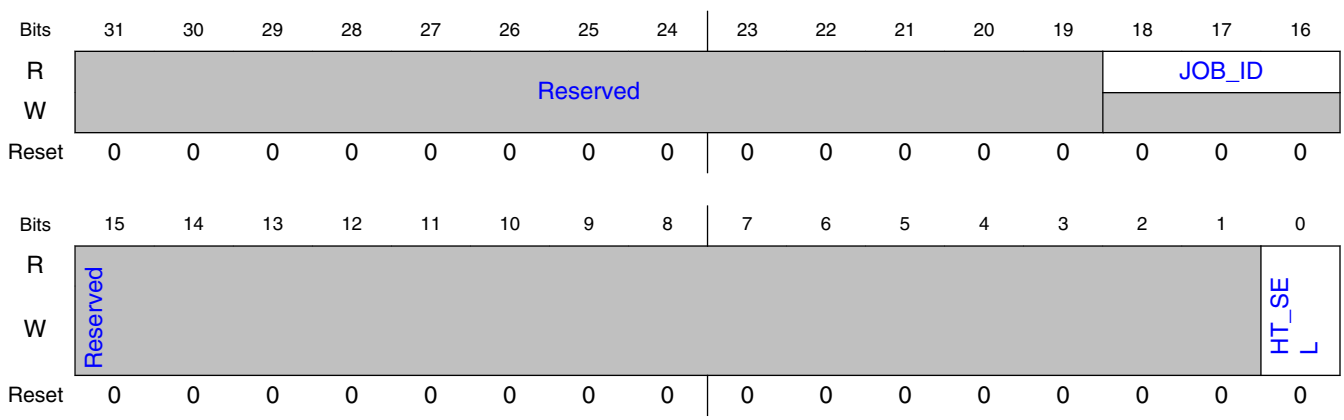
Select register is also used to select the ID of the job that is being queried in the Job Ring Job-Done Source and Job Ring Job-Done Descriptor Address registers. Finally, it specifies which FIFO index to report in the Job Ring Job-Done Job ID FIFO register.

If the value written to the HT_SEL field is larger than the number of holding tanks in CAAM, a value of 0 will be stored in the HT_SEL field and Holding Tank 0 will be used by the HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers.

10.13.104.1 Offset

Register	Offset
JQ_DEBUG_SEL	C24h

10.13.104.2 Diagram



10.13.104.3 Fields

Field	Description
31-19 —	Reserved
18-16 JOB_ID	Job ID. Specifies a Job ID for which to return a Job Source in the Job Ring Job-Done Source FIFO register or Descriptor address in the Job Ring Job-Done Descriptor Address register. Specifies a FIFO index for the Job ID returned by the Job Ring Job-Done Job IDFIFO register, where a value of 0 indicates the oldest job in the FIFO.
15-1	Reserved

Table continues on the next page...

Field	Description
—	
0 HT_SEL	Holding Tank Select. Selects which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers).

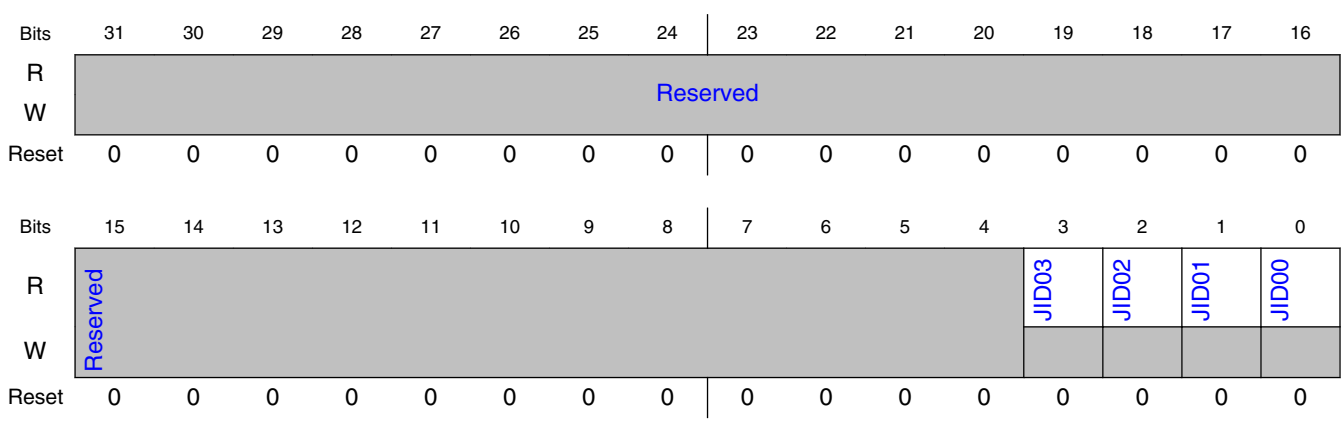
10.13.105 Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS)

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 JRJIDU contains a bit for each of the Job IDs, indicating whether that Job ID is currently in use.

10.13.105.1 Offset

Register	Offset
JRJIDU_LS	DBCh

10.13.105.2 Diagram



10.13.105.3 Fields

Field	Description
31-4 —	Reserved
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.

10.13.106 Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)

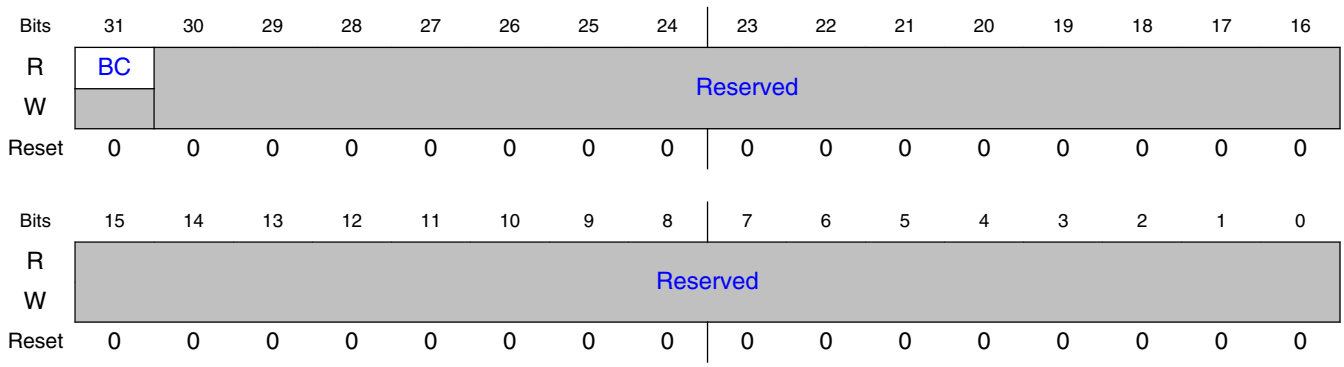
This register indicates whether consistent data has been read from the JRJDJI FIFO and JRJDS and JRJDDA registers.

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 register returns the Job ID located at the index specified in the JOB_ID field of the Job Queue Debug Select Register (JQ_DEBUG_SEL). When the JOB_ID field is set to 0, the oldest JOB_ID in the Job-Done FIFO is returned. See Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR2). 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.

10.13.106.1 Offset

Register	Offset
JRJDJIFBC	DC0h

10.13.106.2 Diagram



10.13.106.3 Fields

Field	Description
31 BC	Been changed. The hardware sets BC to 0 when the job descriptor address for Job ID 0 is read from Job Ring Job-Done Descriptor Address (JRJDDA). The hardware sets BC to 1 when any job is added or removed from the Job-Done Job ID FIFO. After software reads the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA registers software should read BC. If the BC bit is 1, the results read from the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA may be inconsistent with each other.
30-0 —	Reserved

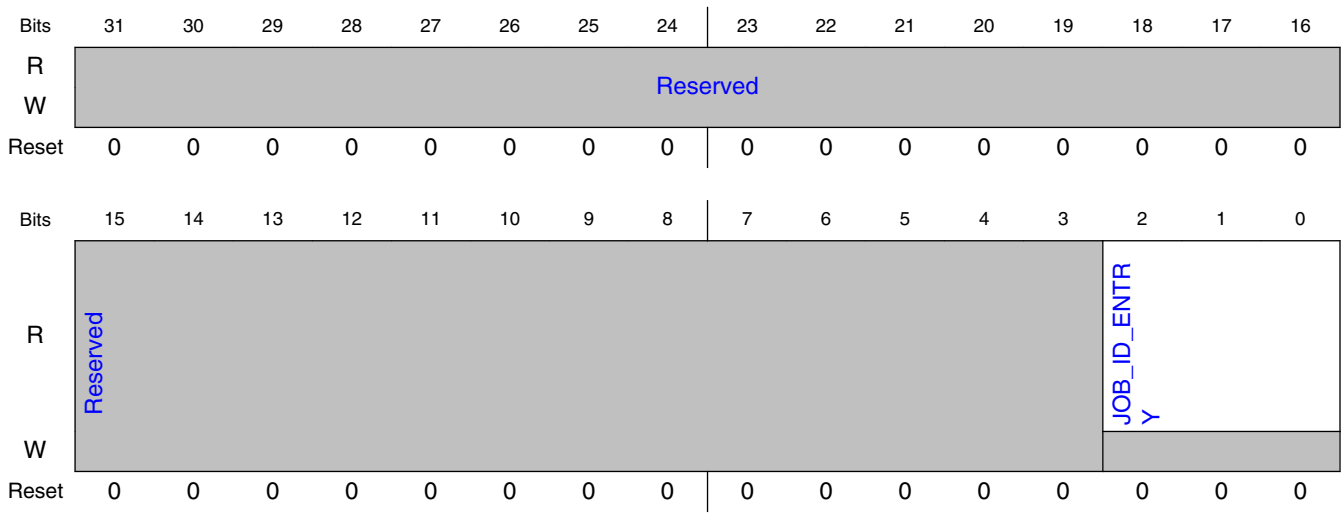
10.13.107 Job Ring Job-Done Job ID FIFO (JRJDJIF)

This register provides Job Ring job ID information required to examine job completion status.

10.13.107.1 Offset

Register	Offset
JRJDJIF	DC4h

10.13.107.2 Diagram



10.13.107.3 Fields

Field	Description
31-3 —	Reserved
2-0 JOB_ID_ENTRY	Job ID entry. This field contains the Job ID of a job whose completion status is located at the JQ_DEBUG_SEL[JOB-ID] index in the Job-Done FIFO.

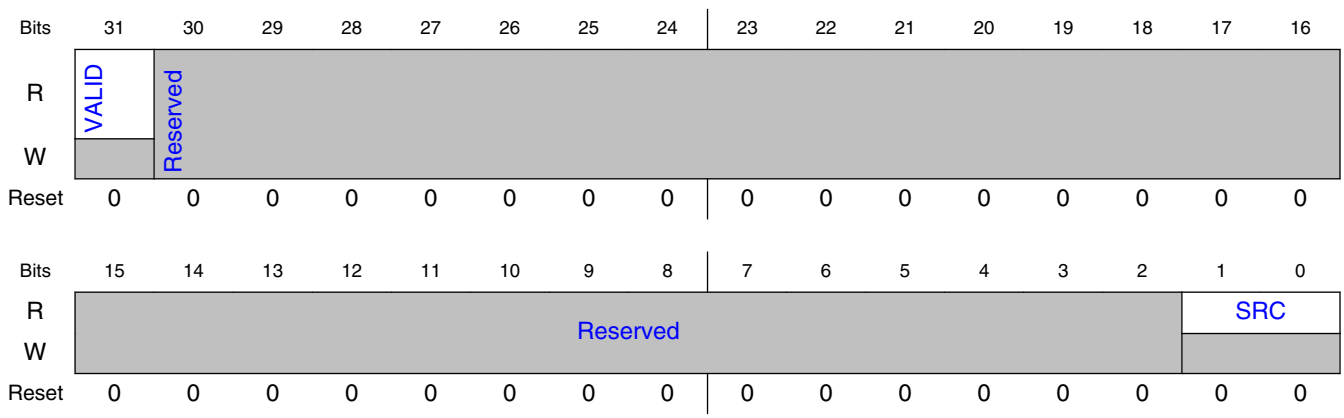
10.13.108 Job Ring Job-Done Source 1 (JRJDS1)

The Job Queue keeps track of the job source (Job Ring numbers) for each Job ID, and values in this register are updated whenever a new Job Ring job starts in a holding tank. Each entry in this register is matched to corresponding entries in the JRJDV and JRDDAa registers.

10.13.108.1 Offset

Register	Offset	Description
JRJDS1	DE4h	The source for the job with the Job ID specified in JQ_DEBUG_SEL[JOB-ID].

10.13.108.2 Diagram



10.13.108.3 Fields

Field	Description
31 VALID	Valid. If this bit is 1, the job located at the index specified by the JOB_ID field in the Job Queue Debug Select register is complete, but its status has not yet been written to the output ring.
30-2 —	Reserved
1-0 SRC	Source. 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. The job is located in the FIFO at the index specified by the JOB_ID field in the Job Queue Debug Select register.

10.13.109 Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)

The JRJDDA register is used to store the address of a job descriptor when the job is sent to a holding tank. The descriptor address read from JRJDDA is the one corresponding to the Job ID specified in the JOB_ID field of the Job Queue Debug Select Register (JQ_DEBUG_SEL). See Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR2). Because these addresses are updated only when a new job starts in a holding tank, some addresses read from this register may be for completed jobs that have already been written to an output ring. This register is intended to be used when debugging descriptor execution via a Job Ring.

10.13.109.1 Offset

Register	Offset	Description
JRJDDA	E00h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.109.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.109.3 Fields

Field	Description
63-32 —	Reserved
31-0 JD_ADDR	Job Descriptor Address.

10.13.110 CHA Revision Number Register, most-significant half (CRNR_MS)

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 CHA Version ID Register, most-significant half (CHAVID_MS)). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

10.13.110.1 Offset

Register	Offset
CRNR_MS	FA0h (alias)
CRNR_MS	1FA0h (alias)
CRNR_MS	2FA0h (alias)
CRNR_MS	3FA0h (alias)
CRNR_MS	6FA0h (alias)
CRNR_MS	8FA0h (alias)

10.13.110.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JRRN				DECORN				Reserved							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ZARN				ZERN				SNW9RN				Reserved			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.110.3 Fields

Field	Description
31-28 JRRN	Job Ring Revision Number
27-24 DECORN	DECO Revision Number
23-16 —	Reserved
15-12 ZARN	ZUC Authentication Hardware Accelerator Revision Number
11-8 ZERN	ZUC Encryption Hardware Accelerator Revision Number
7-4 SNW9RN	SNOW-f9 Hardware Accelerator Revision Number
3-0 —	Reserved

10.13.111 CHA Revision Number Register, least-significant half (CRNR_LS)

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 CHA Version ID Register, most-significant half (CHAVID_MS)). Since the register is larger

than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

10.13.111.1 Offset

Register	Offset
CRNR_LS	FA4h (alias)
CRNR_LS	1FA4h (alias)
CRNR_LS	2FA4h (alias)
CRNR_LS	3FA4h (alias)
CRNR_LS	6FA4h (alias)
CRNR_LS	8FA4h (alias)

10.13.111.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKRN				KASRN				SNW8RN				RNGRN			
W																
Reset	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MDRN				ARC4RN				DESRN				AESRN			
W																
Reset	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0

10.13.111.3 Fields

Field	Description
31-28 PKRN	Public Key Hardware Accelerator Revision Number For PKHA-XT, PKRN=1. For PKHA-SD, see below. 0000 - PKHA-SDv1 0001 - PKHA-SDv2 0010 - PKHA-SDv3 0011 - PKHA-SDv4

Table continues on the next page...

Field	Description
27-24 KASRN	Kasumi f8/f9 Hardware Accelerator Revision Number
23-20 SNW8RN	SNOW-f8 Hardware Accelerator Revision Number
19-16 RNGRN	Random Number Generator Revision Number.
15-12 MDRN	Message Digest Hardware Accelerator module Revision Number.
11-8 ARC4RN	Alleged RC4 Hardware Accelerator Revision Number.
7-4 DESRN	DES Accelerator Revision Number.
3-0 AESRN	AES Accelerator Revision Number. 0000 No Differential Power Analysis resistance implemented 0001 Differential Power Analysis resistance implemented For all other values when AESVID = 4, Differential Power Analysis resistance is implemented.

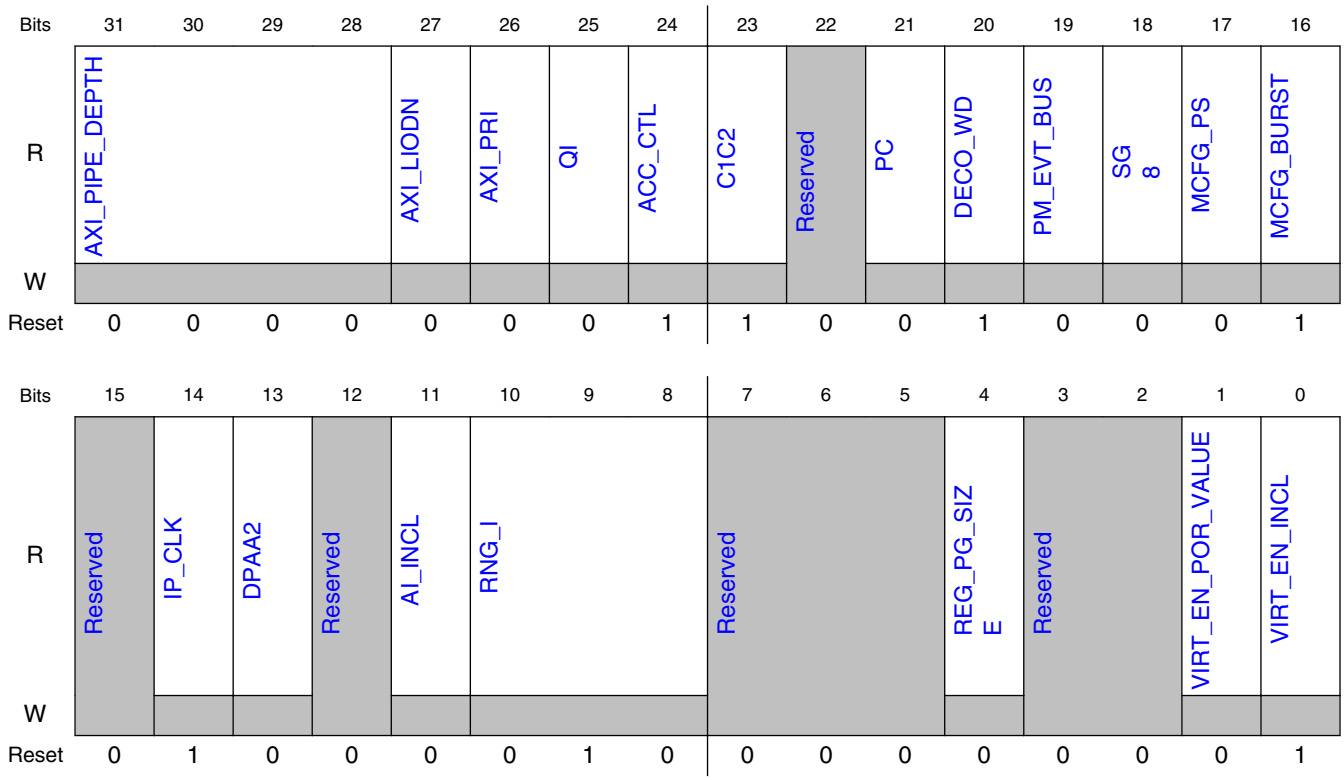
10.13.112 Compile Time Parameters Register, most-significant half (CTPR_MS)

The Compile Time Parameters register indicates the parameter settings at the time CAAM was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.112.1 Offset

Register	Offset
CTPR_MS	FA8h (alias)
CTPR_MS	1FA8h (alias)
CTPR_MS	2FA8h (alias)
CTPR_MS	3FA8h (alias)
CTPR_MS	6FA8h (alias)
CTPR_MS	8FA8h (alias)

10.13.112.2 Diagram



10.13.112.3 Fields

Field	Description
31-28 AXI_PIPE_DEPTH	AXI Pipeline Depth. 0000: CAAM DMA implements maximum AXI bus pipeline depth. 0001 - 1111: CAAM DMA implements an AXI bus pipeline depth as specified by AXI_PIPE_DEPTH.
27 AXI_LIODN	LIODN logic included. 0 : This version of CAAM does **not** implement LIODN logic. 1 : This version of CAAM implements logic to select LIODNs.
26 AXI_PRI	AXI Master Priority implemented. 0 : This version of CAAM does **not** implement AXI Master Priority. 1 : This version of CAAM implements logic for the AXI Master Priority signals.
25 QI	Queue Manager interface (QI) implemented. 0 : This version of CAAM does **not** implement a QI. 1 : This version of CAAM implements a QI.
24	System/user partition-based CAAM IP Bus register access control

Table continues on the next page...

Field	Description
ACC_CTL	0: CAAM does **not** implement partition-based access control for its IP Bus registers, i.e., CAAM relies on the core MMU and/or other SoC register access controls if such control is needed. 1: CAAM implements DID-based IP Bus register access control.
23 C1C2	Separate C1 and C2 registers 0: In this implementation of CAAM the Class 2 Key and Context registers are shared with the Class 1 Key and Context registers. 1: CAAM implements Class 2 Key and Context registers that are separate from the Class 1 Key and Context registers.
22 —	Reserved
21 PC	Performance Counter registers implemented 0: CAAM does **not** implement Performance Counter registers. 1: CAAM does implement Performance Counter registers.
20 DECO_WD	DECO Watchdog Counter implemented 0: CAAM does **not** implement a DECO Watchdog Counter. 1: CAAM does implement a DECO Watchdog Counter.
19 PM_EVT_BUS	Performance Monitor Event Bus implemented 0: CAAM does **not** implement a Performance Monitor Event Bus. 1: CAAM does implement a Performance Monitor Event Bus.
18 SG8	Eight Scatter-Gather Tables implemented 0: CAAM implements one Scatter-Gather Table register. 1: CAAM implements eight Scatter-Gather Table registers.
17 MCFG_PS	Pointer Size field implemented 0: The Master Configuration Register does **not** contain a Pointer Size field. 1: The Master Configuration Register does contain a Pointer Size field.
16 MCFG_BURST	Burst Configurability If MCFG_BURST is 0, the normal burst size is limited to 32 bytes and large bursts cannot be enabled. If MCFG_BURST is 1, the normal burst size is either set to 64 bytes or large bursts can be enabled by setting the LARGE_BURST bit in the Master Configuration Register (see MCFGR[LARGE_BURST] for details). 0: Normal bursts are aligned 32-byte transfers and extended bursting cannot be enabled. 1: Normal bursts are aligned 64-byte transfers or extended bursting exceeding the normal burst size can be enabled with the LARGE_BURST enable bit in the Master Configuration register.
15 —	Reserved
14 IP_CLK	IP Bus Slave Clock. 0: The frequency of CAAM's IP Bus Slave Clock is the same as the frequency of CAAM's AXI bus clock. 1: The frequency of CAAM's IP Bus Slave Clock is one-half of the frequency of CAAM's AXI bus clock.
13 DPAA2	ICIDs with AMQs supported. 0: This version of CAAM does **not** support ICIDs with AMQs. 1: This version of CAAM supports ICIDs with AMQs.

Table continues on the next page...

Field	Description
12 —	Reserved
11 AI_INCL	AIOP interface implemented. 0 : This version of CAAM does **not** implement an AIOP interface. 1 : This version of CAAM implements one or more AIOP interfaces.
10-8 RNG_I	RNG Instantiations. RNG_I indicates the number of RNG instantiations that are implemented in the RNG hardware. Note that each instantiation is the data context for an independent RNG stream. The number of hardware RNGs is indicated in the RNGNUM field of the CHANUM register.
7-5 —	Reserved
4 REG_PG_SIZE	CAAM register page size. 0: CAAM uses 4Kbyte register pages. 1: CAAM uses 64Kbyte register pages.
3-2 —	Reserved
1 VIRT_EN_POR_VALUE	Job Ring Virtualization POR state. 0: Job Ring virtualization is **not** enabled at power up. 1: Job Ring virtualization is enabled at power up.
0 VIRT_EN_INCL	Job Ring Virtualization programmable. 0: Job Ring virtualization is always enabled and the Security Configuration register does not contain a VIRT_EN bit. 1: Job Ring virtualization can be programmed to be enabled or disabled by writing to the VIRT_EN bit in the Security Configuration register.

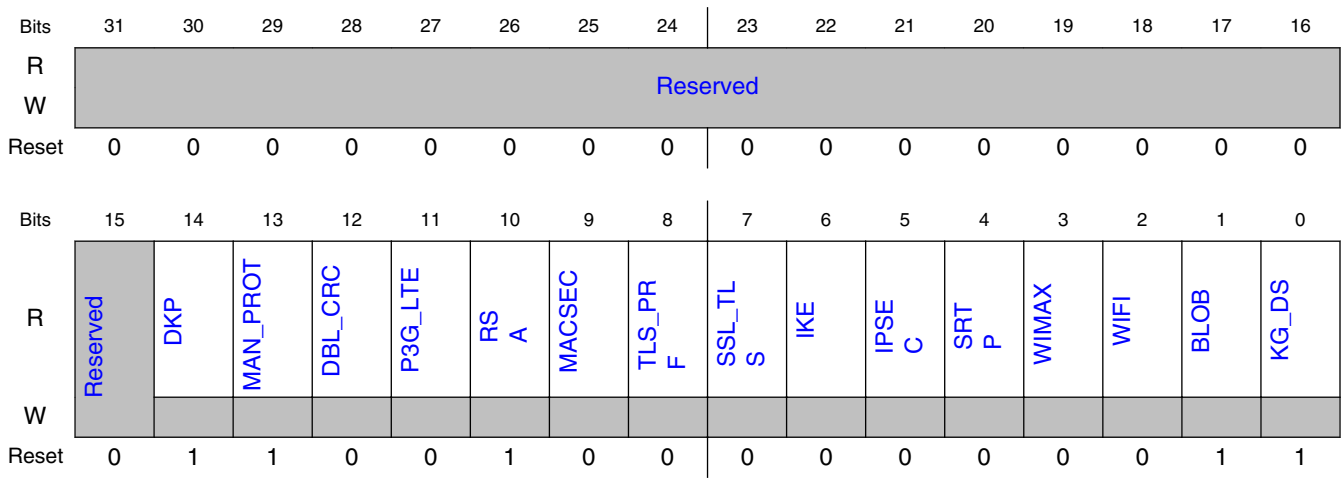
10.13.113 Compile Time Parameters Register, least-significant half (CTPR_LS)

The Compile Time Parameters register indicates the parameter settings at the time CAAM was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.113.1 Offset

Register	Offset
CTPR_LS	FACh (alias)
CTPR_LS	1FACh (alias)
CTPR_LS	2FACh (alias)
CTPR_LS	3FACh (alias)
CTPR_LS	6FACh (alias)
CTPR_LS	8FACh (alias)

10.13.113.2 Diagram



10.13.113.3 Fields

Field	Description
31-16 —	Reserved
15 —	Reserved
14 DKP	Derived Key Protocol 0 - CAAM does not implement the Derived Key Protocol. 1 - CAAM implements the Derived Key Protocol.
13 MAN_PROT	Manufacturing Protection protocol 0 - CAAM does not implement Manufacturing Protection functions.

Table continues on the next page...

Field	Description
	1 - CAAM implements Manufacturing Protection functions.
12 DBL_CRC	Double CRC protocol 0 - CAAM does not implement specialized support for Double CRC. 1 - CAAM implements specialized support for Double CRC.
11 P3G_LTE	3GPP/LTE protocol 0 - CAAM does not implement specialized support for 3G and LTE protocols. 1 - CAAM implements specialized support for 3G and LTE protocols.
10 RSA	RSA protocol 0 - CAAM does not implement specialized support for RSA encrypt and decrypt operations. 1 - CAAM implements specialized support for RSA encrypt and decrypt operations.
9 MACSEC	MACSEC protocol 0 - CAAM does not implement specialized support for the MACSEC protocol. 1 - CAAM implements specialized support for the MACSEC protocol.
8 TLS_PRF	TLS PRF protocol 0 - CAAM does not implement specialized support for the TLS protocol pseudo-random function. 1 - CAAM implements specialized support for the TLS protocol pseudo-random function.
7 SSL_TLS	SSL/TLS protocol 0 - CAAM does not implement specialized support for the SSL and TLS protocols. 1 - CAAM implements specialized support for the SSL and TLS protocols.
6 IKE	IKE protocols 0 - CAAM does not implement specialized support for the IKE protocol. 1 - CAAM implements specialized support for the IKE protocol.
5 IPSEC	IPSEC protocols 0 - CAAM does not implement specialized support for the IPSEC protocol. 1 - CAAM implements specialized support for the IPSEC protocol.
4 SRTP	SRTP protocol 0 - CAAM does not implement specialized support for the SRTP protocol. 1 - CAAM implements specialized support for the SRTP protocol.
3 WIMAX	WiMax protocol 0 - CAAM does not implement specialized support for the WIMAX protocol. 1 - CAAM implements specialized support for the WIMAX protocol.
2 WIFI	WiFi protocol 0 - CAAM does not implement specialized support for the WIFI protocol. 1 - CAAM implements specialized support for the WIFI protocol.
1 BLOB	Blob protocol 0 - CAAM does not implement specialized support for encapsulating and decapsulating cryptographic blobs. 1 - CAAM implements specialized support for encapsulating and decapsulating cryptographic blobs.
0 KG_DS	PK generation and digital signature protcols 0 - CAAM does not implement specialized support for Public Key Generation and Digital Signatures.

Field	Description
	1 - CAAM implements specialized support for Public Key Generation and Digital Signatures.

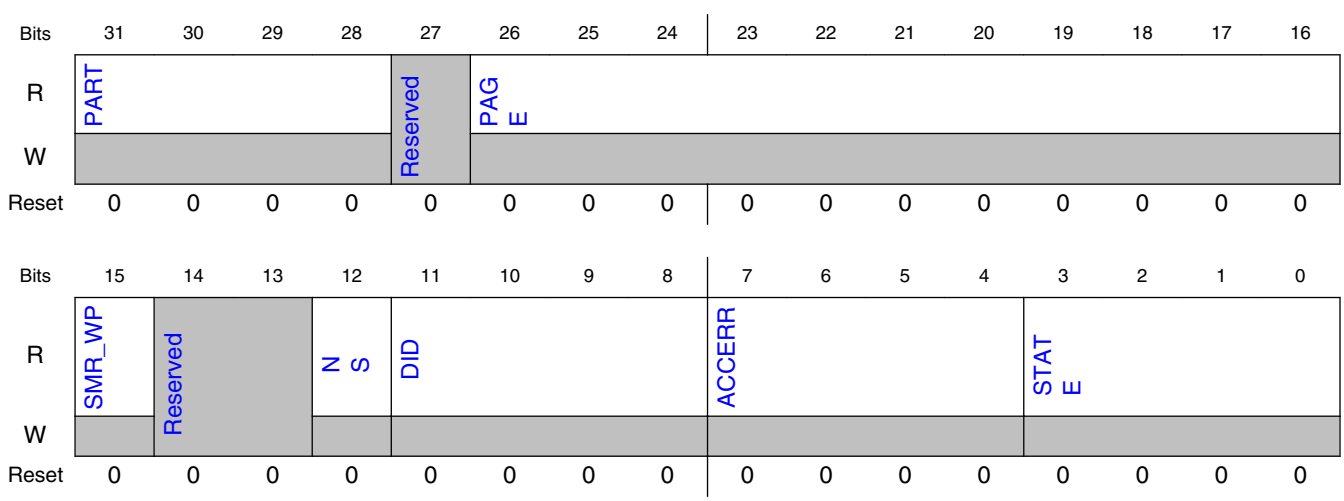
10.13.114 Secure Memory Status Register (SMSTA)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.114.1 Offset

Register	Offset
SMSTA	FB4h (alias)
SMSTA	1FB4h (alias)
SMSTA	2FB4h (alias)
SMSTA	3FB4h (alias)
SMSTA	6FB4h (alias)
SMSTA	8FB4h (alias)

10.13.114.2 Diagram



10.13.114.3 Fields

Field	Description
31-28 PART	In this version of CAAM this field should be ignored. Find out the partition number via the SMCS register's Page Inquiry Command.
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 SMR_WP	Secure Memory Registers Write Protected. SMR_WP=1 indicates that the CAAM manager has disabled writing into the Secure Memory registers in this Job Ring page.
14-13 —	Reserved
12 NS	The TrustZone nonsecure bit of the bus master whose access to Secure Memory was denied. NS=0 means SecureWorld.
11-8 DID	The DID 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. 0000 - No error occurred 0001 - A bus transaction attempted to access a page in Secure Memory, but the page was not allocated to any partition. 0010 - A bus transaction attempted to access a partition, but the transaction's TrustZone World, DID was not granted access to the partition in the partition's SMAG2/1JR registers. 0011 - A bus transaction attempted to read, but reads from this partition are not allowed. 0100 - A bus transaction attempted to write, but writes to this partition are not allowed. 0110 - Secure Memory Blob import or export was attempted, but Secure Memory Blob access is not allowed for this partition. 1010 - A Descriptor attempted a Secure Memory Blob import or export, but not all of the pages referenced were from the same partition. 1011 - 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 4kbyte address block occupied by a 1Kbyte or 2Kbyte Secure Memory page. 1100 - A bus transaction was attempted, but the burst would have crossed a page boundary. 1101 - An attempt was made to access a page while it was still being initialized.
3-0 STATE	Current State. This field represents the current state of the Secure Memory Controller. 0000 - Reset State 0001 - Initialize State 0010 - Normal State 0011 - Fail State

10.13.115 Secure Memory Partition Owners 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's SDID owns the partition, another Job Ring's SDID owns the partition, the partition is unowned, or the partition is unimplemented. Each allocated partition is allocated to one of the 4096 TrustZone SecureWorld SDID values or to one of the 4096 TrustZone NonSecureWorld SDID values. A partition owner may own more than one partition.

Note that the SMPO is accessible via multiple address aliases (one per Job Ring plus one located in register page 0). If Job Ring x's owner[15] (as determined by the JRaDID register) reads the SMPO register via the address associated with Job Ring x, then the partition owner field will return a 11b code for each partition owned by that Job Ring. If the partition is owned by a different owner, a 10b code will be returned in the partition's partition owner field. If the partition is currently unallocated a 00b code will be returned in the partition owner field. Note that the same entity can own more than one Job Ring, and can obtain the same information by reading the SMPO register from different address aliases.

Note that at POR all Secure Memory pages are allocated to partition 0, and partition 0 is assigned to SDID value 0000h. Since all JRDID registers also reset to 0000h, at POR all Job Rings own partition 0. Consequently, if any Job Ring alias of SMPO is read immediately following POR, or if the SMPO alias in register page 0 immediately following POR, PO0 will return 11b and all other partition owner fields will return either 00b (unowned) or 01b (unimplemented). If the SMPO is read from a Job Ring alias after the Job Ring has been assigned a new SDID value, or if the SDID value in the PAGE0_SDID register has been changed, PO0 will return 10b (owned by someone else) and all other partition owner fields will return either 00b (unowned) or 01b (unimplemented).

10.13.115.1 Offset

Register	Offset	Description
SMPO	FBCh (alias)	used by the hypervisor or Trustzone SecureWorld
SMPO	1FBCh (alias)	used by Job Ring 0

Table continues on the next page...

15. Bit 15 of the PSDIDR is ignored when determining read/write access to the partition's SMAP and SMAG registers or when reading the SMPO register.

Register	Offset	Description
SMPO	2FBCh (alias)	used by Job Ring 1
SMPO	3FBCh (alias)	used by Job Ring 2

10.13.115.2 Diagram

Bits	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																
Reset	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Bits	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

10.13.115.3 Fields

Field	Description
31-30 PO15	Partition Owner for partition 15. See PO0.
29-28 PO14	Partition Owner for partition 14. See PO0.
27-26 PO13	Partition Owner for partition 13. See PO0.
25-24 PO12	Partition Owner for partition 12. See PO0.
23-22 PO11	Partition Owner for partition 11. See PO0.
21-20 PO10	Partition Owner for partition 10. See PO0.
19-18 PO9	Partition Owner for partition 9. See PO0.
17-16 PO8	Partition Owner for partition 8. See PO0.
15-14 PO7	Partition Owner for partition 7. See PO0.

Table continues on the next page...

Field	Description
13-12 PO6	Partition Owner for partition 6. See PO0.
11-10 PO5	Partition Owner for partition 5. See PO0.
9-8 PO4	Partition Owner for partition 4. See PO0.
7-6 PO3	Partition Owner for partition 3. See PO0.
5-4 PO2	Partition Owner for partition 2. See PO0.
3-2 PO1	Partition Owner for partition 1. See PO0.
1-0 PO0	Partition Owner for partition 0: When read by a Job Ring owner, this field indicates if partition 0 is owned by that Job Ring, another Job Ring, Unowned, or Unimplemented. 00 - Available; Unowned. A Job Ring owner may claim partition 0 by writing to the appropriate SMAPJR 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 0 does not exist in this version 10 - Another entity owns partition 0. Partition 0 is unavailable to the reader. If the reader attempts to de-allocate partition 0 or write to the SMAPJR register or SMAGJR register for partition 0 or allocate a page to or de-allocate a page from partition 0 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 0b10 value for that partition until all its pages have been zeroized.) 11 - The entity that read the SMPO register owns partition 0. Ownership is claimed when the access permissions register (SMAPJR) of an available partition is first written.

10.13.116 Fault Address Register (FAR)

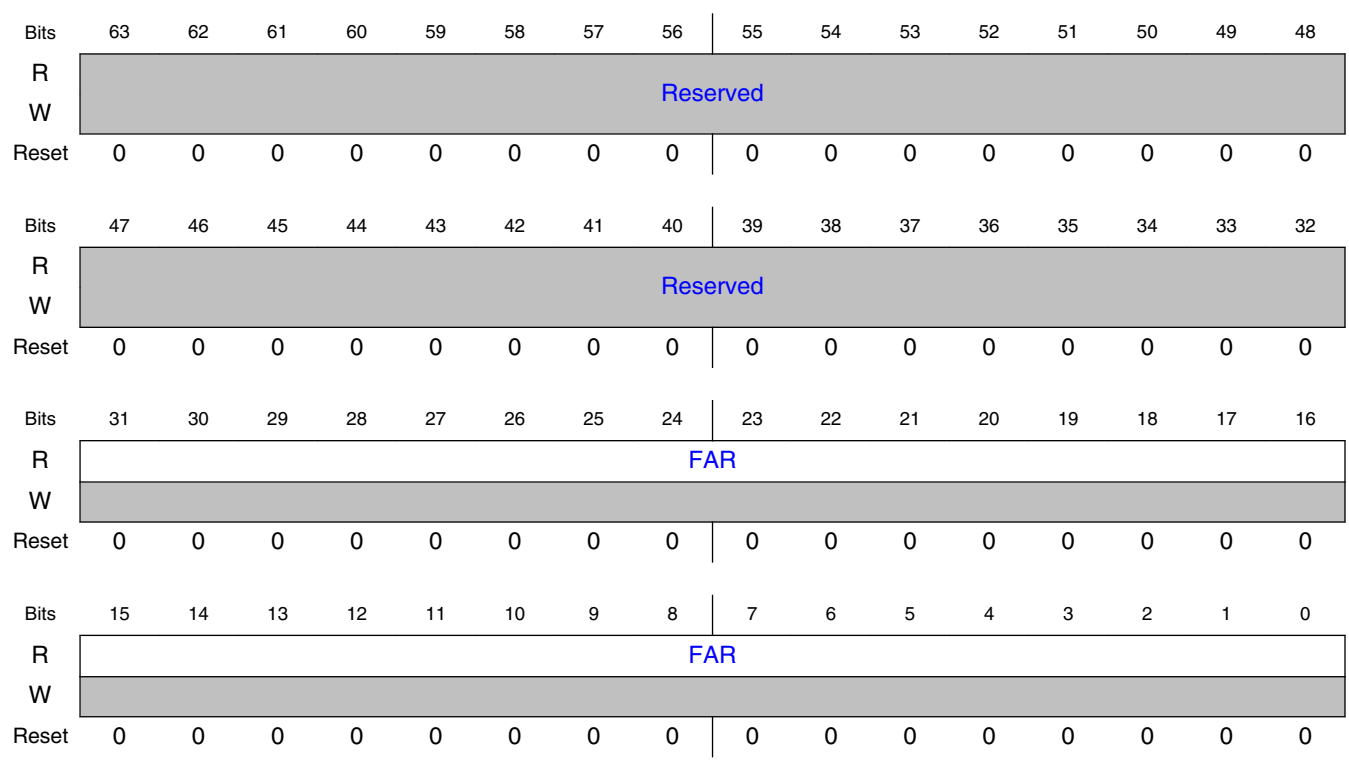
The Fault Address Register is used for software debugging of external memory access errors. This register will hold the value of the AXI address where a read or write error occurred. The read error address is aligned to the data bus address boundary of the data sample where the error occurred. The write 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 Fault Address Detail Register (FADR)). The associated DID is in the Fault Address DID Register (see Section Fault Address DID Register (FADID)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces. The values in the

Fault Address Register, the Fault Address DID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers have been read, in any order, whereupon all these registers will be cleared.

10.13.116.1 Offset

Register	Offset	Description
FAR	FC0h (alias)	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).
FAR	1FC0h (alias)	-
FAR	2FC0h (alias)	-
FAR	3FC0h (alias)	-
FAR	6FC0h (alias)	-
FAR	8FC0h (alias)	-

10.13.116.2 Diagram



10.13.116.3 Fields

Field	Description
63-32 —	Reserved
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.

10.13.117 Fault Address DID Register (FADID)

The Fault Address DID Register is used by software for debugging external memory access errors. This register indicates the DID associated with the AXI transaction where the error occurred. The associated AXI address is in the Fault Address Register (see Fault Address Register (FAR)) and additional details appear in the Fault Address Detail Register (see Fault Address Detail Register (FADR)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces. The values in the Fault Address Register, the Fault Address DID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers have been read, in any order, whereupon all these registers will be cleared.

10.13.117.1 Offset

Register	Offset
FADID	FC8h (alias)
FADID	1FC8h (alias)
FADID	2FC8h (alias)
FADID	3FC8h (alias)
FADID	6FC8h (alias)
FADID	8FC8h (alias)

10.13.117.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FICID											FNS	FDID			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.117.3 Fields

Field	Description
31-16 —	Reserved
15-5 FICID	DMA transaction ICID. This was the ICID value associated with the DMA transaction that failed.
4 FNS	DMA transaction ns. This was the ns signal (ns=0 means TrustZone SecureWorld) associated with the DMA transaction that failed.
3-0 FDID	DMA transaction DID. This was the DID associated with the DMA transaction that failed.

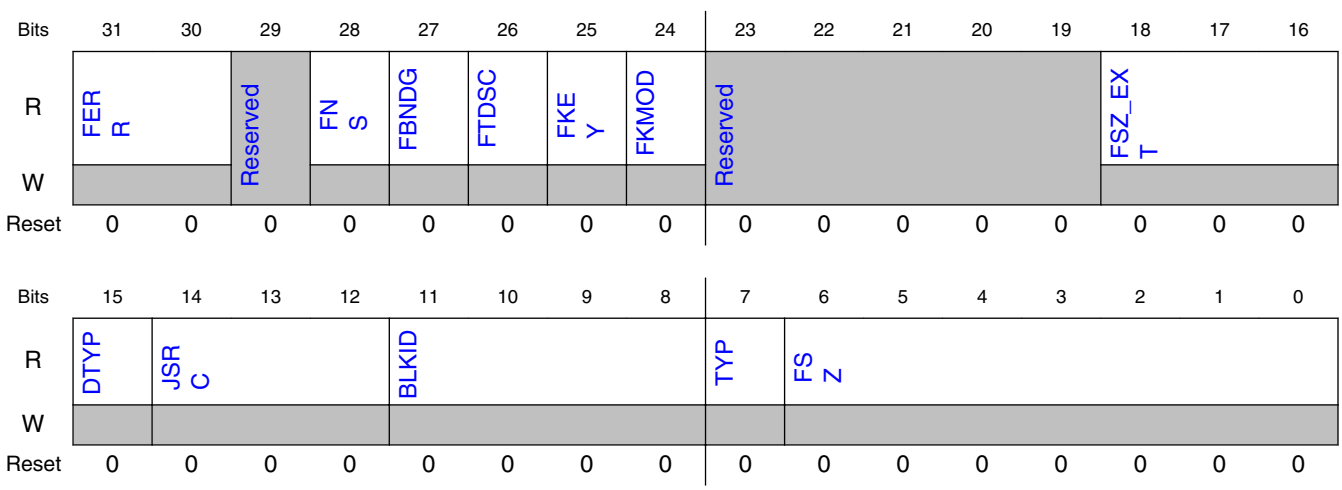
10.13.118 Fault Address Detail Register (FADR)

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 (FAR). The associated DID is in the Fault Address DID Register (FADID). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces. The values in the Fault Address Register, the Fault Address DID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers have been read, in any order, whereupon all these registers will be cleared.

10.13.118.1 Offset

Register	Offset
FADR	FCCh (alias)
FADR	1FCCh (alias)
FADR	2FCCh (alias)
FADR	3FCCh (alias)
FADR	6FCCh (alias)
FADR	8FCCh (alias)

10.13.118.2 Diagram



10.13.118.3 Fields

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
28	Non-secure (AXI bus ns bit) access to Secure Memory.

Table continues on the next page...

Field	Description
FNS	0 - CAAM DMA was asserting ns=0 at the time of the DMA error. 1 - CAAM DMA was asserting ns=1 at the time of the DMA error.
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 APJR 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
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
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 - Job Ring 2 011 - reserved 100 - RTIC 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: 0100 - job queue controller Burst Buffer 0101 - One of the Job Rings (see JSRC field) 1000 - DECO0

Table continues on the next page...

Field	Description
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.

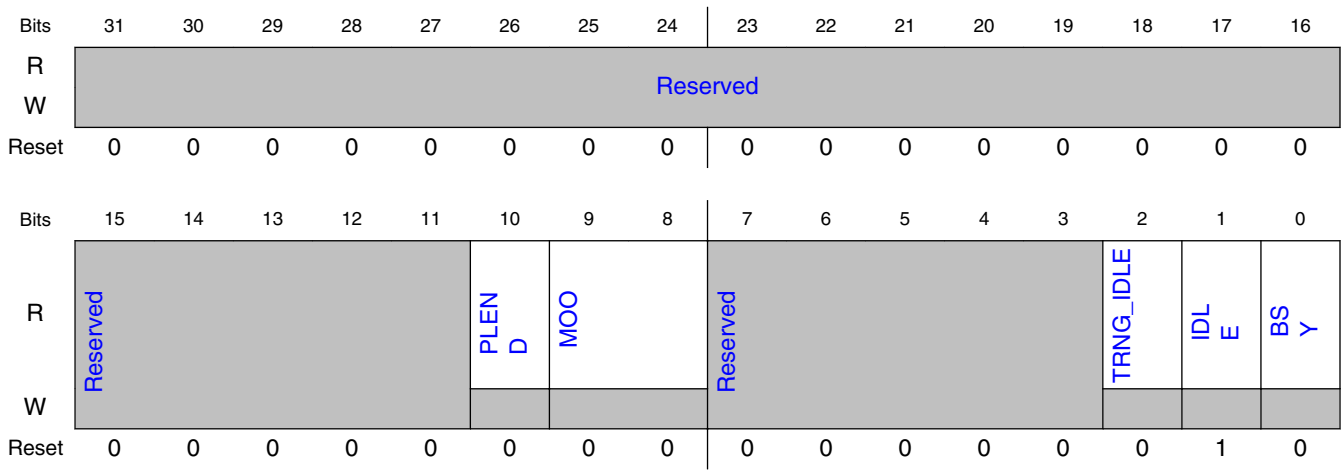
10.13.119 CAAM Status Register (CSTA)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.119.1 Offset

Register	Offset
CSTA	FD4h (alias)
CSTA	1FD4h (alias)
CSTA	2FD4h (alias)
CSTA	3FD4h (alias)
CSTA	6FD4h (alias)
CSTA	8FD4h (alias)

10.13.119.2 Diagram



10.13.119.3 Fields

Field	Description
31-11 —	Reserved
10 PLEND	Platform Endianness.The value of PLEND indicates whether CAAM interprets and generates multi-byte structures in platform memory by default in big endian (BE) or little endian (LE) format. **NOTE:** In this instance of CAAM the PLEND value is a hard-wired, SoC-specific CAAM configuration and software can override the endianess for CAAM sub-module specific data by setting bits in the Job Ring Configuration Register, the RTIC Endian 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 Platform Security State Machine. The modes are defined in CAAM modes of operation. 00 - Non-Secure 01 - Secure 10 - Trusted 11 - Fail
7-3 —	Reserved
2 TRNG_IDLE	TRNG Idle. If TRNG_IDLE == 1, the TRNG portion of the RNG is idle. The free-running oscillator is stopped, so no entropy is being generated.
1 IDLE	CAAM Idle. IDLE == 1 indicates that • CAAM is not busy (see BSY bit description below)

Table continues on the next page...

Field	Description
	• the output job ring timers are not counting, i.e., there are no completed jobs yet to be reported via interrupt (though there may be job entries in output rings that have beed previously reported as complete, but not yet consumed by software) • there are no pending CAAM interrupts (or all such interrupts are masked) • secure memory is idle • and the special RTIC conditions described below are met: If RIDLE == 1 in the RTIC Control Register, IDLE will be 0 if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits are 1). If RIDLE == 0 and CAAM is otherwise idle, the IDLE bit will still occasionally be 0 while RTIC is actually hashing a chunk of memory. That is, if RTIC is in Run-Time Mode and one or more memory blocks are enabled, RTIC's Programmable DMA Throttle Timer may time out periodically and RTIC will launch a hashing job, which will cause IDLE to briefly go to 0.
0 BSY	CAAM Busy. BSY == 1 indicates that CAAM is processing at least one job descriptor in any of CAAM's service interfaces or an external RNG request is being processed.

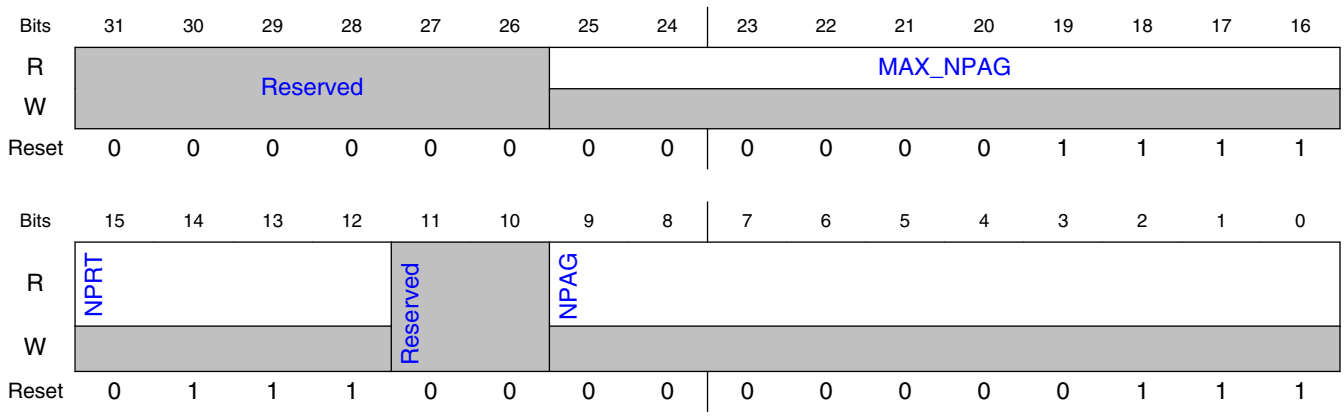
10.13.120 Secure Memory Version ID Register, most-significant half (SMVID_MS)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces. Since the register holds more than 32 bits, it is accessed as two 32-bit words.

10.13.120.1 Offset

Register	Offset
SMVID_MS	FD8h (alias)
SMVID_MS	1FD8h (alias)
SMVID_MS	2FD8h (alias)
SMVID_MS	3FD8h (alias)
SMVID_MS	6FD8h (alias)
SMVID_MS	8FD8h (alias)

10.13.120.2 Diagram



10.13.120.3 Fields

Field	Description
31-26 —	Reserved
25-16 MAX_NPAG	Maximum allowable value for NPAG. Since NPAG counts from 0, the maximum number of Secure Memory pages that can be supported in this version of the CAAM hardware is MAX_NPAG + 1. The NPAG field indicates the highest numbered Secure Memory page that has been configured in this SOC. Since this SoC may use a smaller Secure Memory RAM than the maximum that can be supported by CAAM, the actual number of pages (NPAG + 1) may be smaller than MAX_NPAG + 1.
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.

10.13.121 Secure Memory Version ID Register, least-significant half (SMVID_LS)

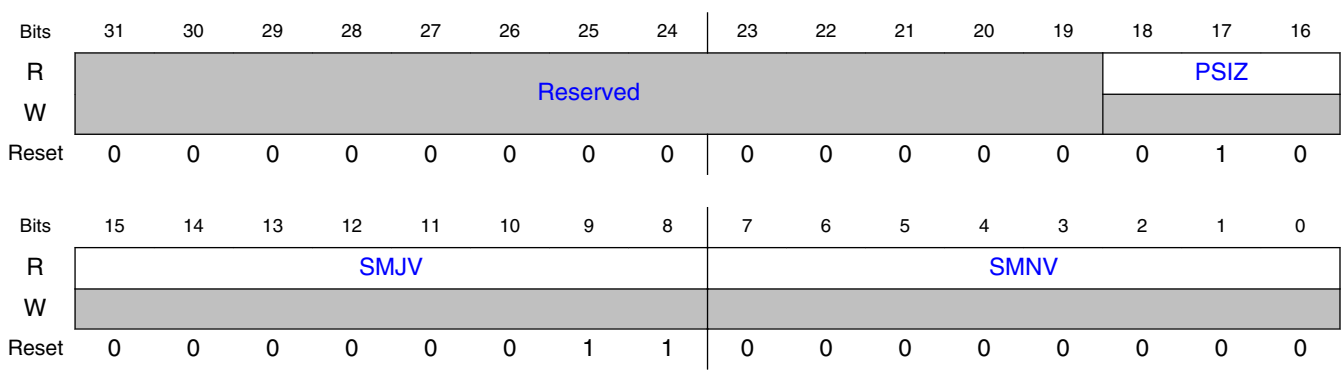
The Secure Memory Version ID register can be used by software to differentiate between different versions of the Secure Memory, and to determine the page size and the number of partitions and pages supported by this version of CAAM. Because this register may be

of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces. Since the register holds more than 32 bits, it is accessed as two 32-bit words.

10.13.121.1 Offset

Register	Offset
SMVID_LS	FDCh (alias)
SMVID_LS	1FDCh (alias)
SMVID_LS	2FDCh (alias)
SMVID_LS	3FDCh (alias)
SMVID_LS	6FDCh (alias)
SMVID_LS	8FDCh (alias)

10.13.121.2 Diagram



10.13.121.3 Fields

Field	Description
31-19 —	Reserved
18-16 PSIZ	Page Size. The number of bytes in a Secure Memory page is 2^{psiz} KB. Zero means 1kbyte 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.

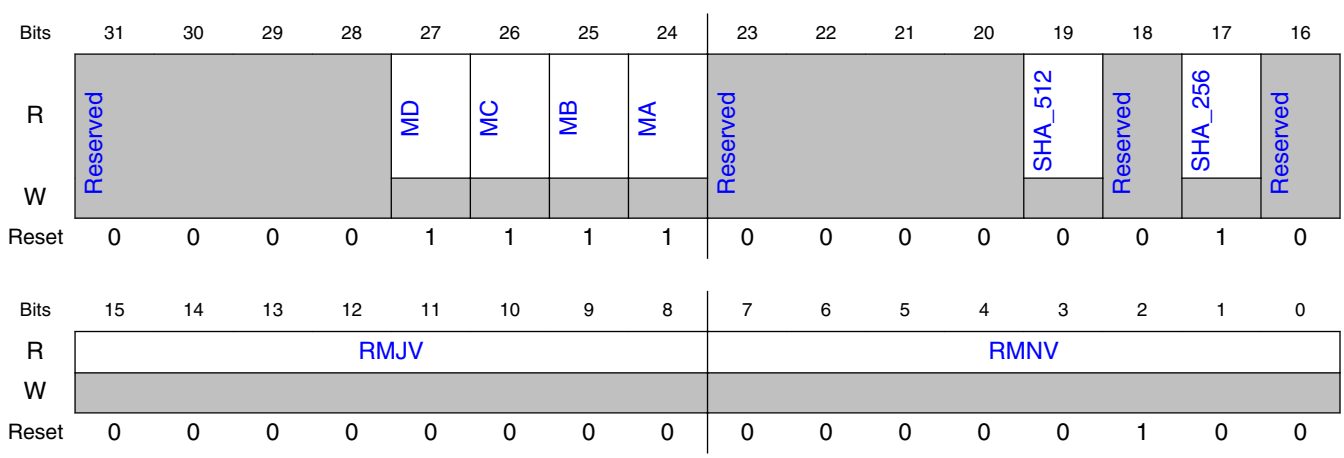
10.13.122 RTIC Version ID Register (RVID)

The Run Time Integrity Checking Version ID register can be used by software to differentiate between different versions of the RTIC. Field RMJV is used for major revisions, field RMNV is used for minor revisions and the remaining fields are used for other revision information about the hardware. The bit assignments of this register appear below. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.122.1 Offset

Register	Offset
RVID	FE0h (alias)
RVID	1FE0h (alias)
RVID	2FE0h (alias)
RVID	3FE0h (alias)
RVID	6FE0h (alias)
RVID	8FE0h (alias)

10.13.122.2 Diagram



10.13.122.3 Fields

Field	Description
31-28 —	Reserved
27 MD	Memory Block D Available. This bit indicates that Memory Block D is available for Hash Once and Run Time Checking.
26 MC	Memory Block C Available. This bit indicates that Memory Block C is available for Hash Once and Run Time Checking.
25 MB	Memory Block B Available. This bit indicates that Memory Block B is available for Hash Once and Run Time Checking.
24 MA	Memory Block A Available. This bit indicates that Memory Block A is available for Hash Once and Run Time Checking.
23-20 —	Reserved
19 SHA_512	SHA-512. 0 - RTIC cannot use the SHA-512 hashing algorithm. 1 - RTIC can use the SHA-512 hashing algorithm.
18 —	Reserved
17 SHA_256	SHA-256. 0 - RTIC cannot use the SHA-256 hashing algorithm. 1 - RTIC can use the SHA-256 hashing algorithm.
16 —	Reserved
15-8 RMJV	RTIC Major Version. Represents major revision number of RTIC. This value is incremented when major functional changes are introduced or the programming model has changed.
7-0 RMNV	RTIC Minor Version. Represents minor revision number of RTIC. This value is incremented when minor functional changes are made that do not change the programming model. Corrections that require changes to the design are the typical reason for incrementing these bits.

10.13.123 CHA Cluster Block 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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.123.1 Offset

Register	Offset
CCBVID	FE4h (alias)
CCBVID	1FE4h (alias)
CCBVID	2FE4h (alias)
CCBVID	3FE4h (alias)
CCBVID	6FE4h (alias)
CCBVID	8FE4h (alias)

10.13.123.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CAAM_ERA								Reserved							
W																
Reset	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AMJV								AMNV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

10.13.123.3 Fields

Field	Description
31-24 CAAM_ERA	CAAM Era. This version of CAAM is based on Era 9 RTL. **NOTE:** A value of 0 implies CAAM Era 5 or earlier.
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.

10.13.124 CHA Version ID Register, most-significant half (CHAVID_MS)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.124.1 Offset

Register	Offset
CHAVID_MS	FE8h (alias)
CHAVID_MS	1FE8h (alias)
CHAVID_MS	2FE8h (alias)
CHAVID_MS	3FE8h (alias)
CHAVID_MS	6FE8h (alias)
CHAVID_MS	8FE8h (alias)

10.13.124.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JRVID				DECOVID				Reserved							
W																
Reset	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ZAVID				ZEVID				SNW9VID				Reserved			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.124.3 Fields

Field	Description
31-28	Job Ring Version ID

Table continues on the next page...

Field	Description
JRVID	
27-24 DECOVID	DECO Version ID
23-16 —	Reserved
15-12 ZAVID	ZUC Authentication Hardware Accelerator Version ID
11-8 ZEVID	ZUC Encryption Hardware Accelerator Version ID
7-4 SNW9VID	SNOW-f9 Hardware Accelerator Version ID
3-0 —	Reserved

10.13.125 CHA Version ID Register, least-significant half (CHAVID_LS)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.125.1 Offset

Register	Offset
CHAVID_LS	FECh (alias)
CHAVID_LS	1FECh (alias)
CHAVID_LS	2FECh (alias)
CHAVID_LS	3FECh (alias)
CHAVID_LS	6FECh (alias)
CHAVID_LS	8FECh (alias)

10.13.125.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKVID				KASVID				SNW8VID				RNGVID			
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MDVID				ARC4VID				DESVID				AESVID			
W																
Reset	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	1

10.13.125.3 Fields

Field	Description
31-28 PKVID	Public Key Hardware Accelerator Version ID. The bit count is the size of the digit used during computation. The single-digit ("SD") versions allow a minimum modulus size of one byte. 0000 - PKHA-XT (32-bit); minimum modulus five bytes 0001 - PKHA-SD (32-bit) 0010 - PKHA-SD (64-bit) 0011 - PKHA-SD (128-bit)
27-24 KASVID	Kasumi f8/f9 Hardware Accelerator Version ID.
23-20 SNW8VID	SNOW-f8 Hardware Accelerator Version ID.
19-16 RNGVID	Random Number Generator Version ID. 0010 - RNGB 0100 - RNG4
15-12 MDVID	Message Digest Hardware Accelerator Version ID. 0000 - Low-power MDHA, with SHA-1, SHA-256, SHA 224, MD5 and HMAC 0001 - Low-power MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5 and HMAC 0010 - Medium-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC 0011 - High-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC
11-8 ARC4VID	Alleged RC4 Hardware Accelerator Version ID. 0000 - Low-power ARC4 0001 - High-performance ARC4
7-4	DES Accelerator Version ID.

Table continues on the next page...

Field	Description
DESVID	
3-0 AESVID	AES Accelerator Version ID. 0011 - Low-power AESA, implementing ECB, CBC, CBC-CS2, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, and GCM modes 0100 - High-performance AESA, implementing ECB, CBC, CBC-CS2, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, CBCXCBC, CTRXCBC, XTS, and GCM modes

10.13.126 CHA Number Register, most-significant half (CHANUM_MS)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.126.1 Offset

Register	Offset
CHANUM_MS	FF0h (alias)
CHANUM_MS	1FF0h (alias)
CHANUM_MS	2FF0h (alias)
CHANUM_MS	3FF0h (alias)
CHANUM_MS	6FF0h (alias)
CHANUM_MS	8FF0h (alias)

10.13.126.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JRNUM				DECONUM				Reserved							
W																
Reset	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ZANUM				ZENUM				SNW9NUM				CRCNUM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.126.3 Fields

Field	Description
31-28 JRNUM	The number of copies of the Job Ring that are implemented in this version of CAAM
27-24 DECONUM	The number of copies of the DECO that are implemented in this version of CAAM
23-16 —	Reserved
15-12 ZANUM	The number of copies of ZUCA that are implemented in this version of CAAM
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

10.13.127 CHA Number Register, least-significant half (CHAN UM_LS)

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 software entities, this register is aliased to addresses in multiple 4kbyte address spaces.

10.13.127.1 Offset

Register	Offset
CHANUM_LS	FF4h (alias)
CHANUM_LS	1FF4h (alias)
CHANUM_LS	2FF4h (alias)
CHANUM_LS	3FF4h (alias)
CHANUM_LS	6FF4h (alias)
CHANUM_LS	8FF4h (alias)

10.13.127.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKNUM				KASNUM				SNW8NUM				RNGNUM			
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MDNUM				ARC4NUM				DESNUM				AESNUM			
W																
Reset	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

10.13.127.3 Fields

Field	Description
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	The number of copies of the SNOW-f8 module that are implemented in this version of CAAM

Table continues on the next page...

Field	Description
SNW8NUM	
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 DESNUM	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.

10.13.128 Input Ring Base Address Register for Job Ring a (IRBAR_JR0 - IRBAR_JR2)

The Input Ring Base Address register holds the physical address of the input ring in memory (see Address pointers. Because there are 3 Job Rings, there are 3 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRBAR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers. 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 or size 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. More information on job management can be found in Job Ring interface.

10.13.128.1 Offset

Register	Offset	Description
IRBAR_JR0	1000h	Used by JR0. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR0DID register
IRBAR_JR1	2000h	Used by JR1. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR1DID register
IRBAR_JR2	3000h	Used by JR2. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR2DID register

10.13.128.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IRBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IRBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.128.3 Fields

Field	Description
63-32 —	Reserved
31-0 IRBA	Input Ring Base Address.

10.13.129 Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR2)

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 3 Job Rings, there are 3 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRSR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers. This 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 or size invalid write error (type 5h) 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 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. See Address pointers for a discussion of address pointers. More information on job management can be found in Job Ring interface.

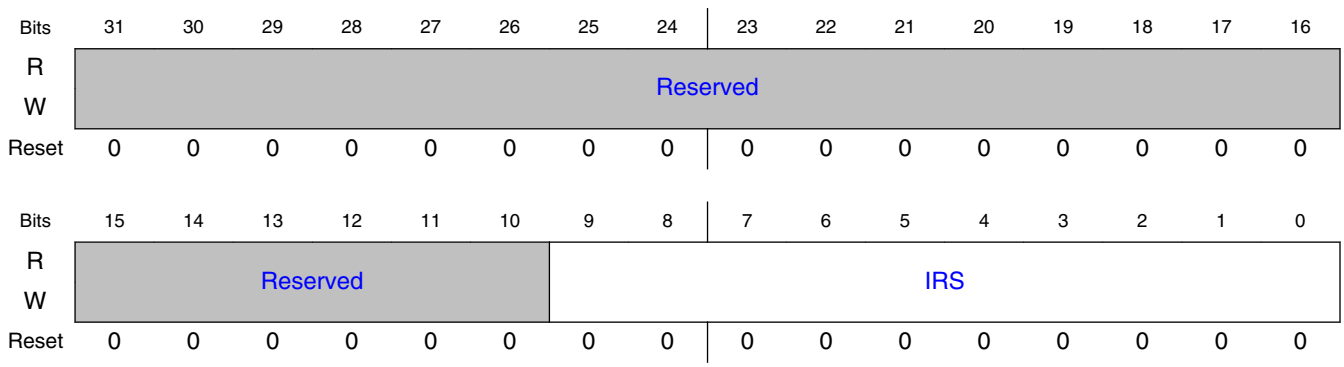
10.13.129.1 Offset

Register	Offset	Description
IRSR_JR0	100Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
IRSR_JR1	200Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register

Table continues on the next page...

Register	Offset	Description
IRSR_JR2	300Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.129.2 Diagram



10.13.129.3 Fields

Field	Description
31-10 —	Reserved
9-0 IRS	Input Ring Size. (measured in number of entries)

10.13.130 Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR2)

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 3 Job Rings, there are 3 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 Input Ring Jobs Added Register for Job Ringa (IRJAR_JR0 - IRJAR_JR2)) when that register is updated. The value of the Input Ring Slots Available

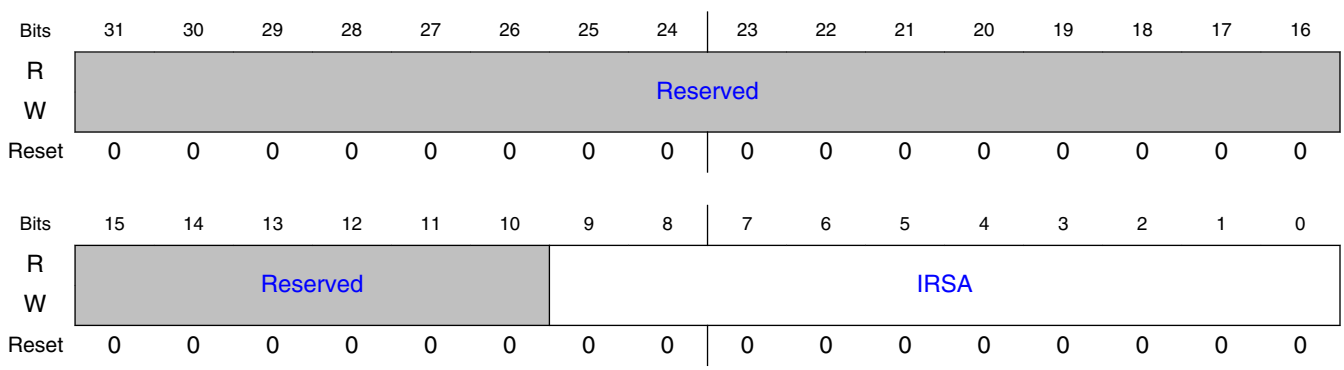
Register will never be larger than the Input Ring Size Register (see Section Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR2)). More information on job management can be found in Job Ring interface.

The Job Ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, IRSAR may only be written with a transaction with ns=0. See Section Job Ring Registers.

10.13.130.1 Offset

Register	Offset	Description
IRSAR_JR0	1014h	(used by JR 0) Accessible only when using DID specified in JR0DID register
IRSAR_JR1	2014h	(used by JR 1) Accessible only when using DID specified in JR1DID register
IRSAR_JR2	3014h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.130.2 Diagram



10.13.130.3 Fields

Field	Description
31-10 —	Reserved
9-0 IRSA	Input Ring Slots Available. (measured in number of available job slots)

10.13.131 Input Ring Jobs Added Register for Job Ringa (IRJAR_JR0 - IRJAR_JR2)

The Input Ring Jobs Added Register tells CAAM how many new jobs were added to the input ring. Because there are 3 Job Rings, there are 3 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 Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR2)). 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. More information on job management can be found in Job Ring interface.

When the Job Ring is allocated to TrustZone SecureWorld, IRJAR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers.

10.13.131.1 Offset

Register	Offset	Description
IRJAR_JR0	101Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
IRJAR_JR1	201Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
IRJAR_JR2	301Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.131.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved						IRJA									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.131.3 Fields

Field	Description
31-10 —	Reserved
9-0 IRJA	Input Ring Jobs Added. (measured in number of entries)

10.13.132 Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR2)

The Output Ring Base Address Register holds the address of the output ring in memory (see Job Ring interface). Because there are 3 Job Rings, there are 3 copies of this register. When the Job Ring is allocated to TrustZone SecureWorld, ORBAR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers. 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 or size 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. More information on job management can be found in Job Ring interface.

10.13.132.1 Offset

Register	Offset	Description
ORBAR_JR0	1020h	Used by JR0. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR0DID register
ORBAR_JR1	2020h	Used by JR1. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR1DID register
ORBAR_JR2	3020h	Used by JR2. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).Accessible only when using DID specified in JR2DID register

10.13.132.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ORBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ORBA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.132.3 Fields

Field	Description
63-32 —	Reserved
31-0 ORBA	Output Ring Base Address.

10.13.133 Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR2)

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 section

Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR2_MS)). See Address pointers for a discussion of address pointers. Because there are 3 Job Rings, there are 3 copies of this register. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers. This register can be written only when the Job Ring is halted or when there are no jobs from this ring in the input ring or output ring or in progress within CAAM, else an *output ring base address or size 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 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.

More information on job management can be found in Job Ring interface.

10.13.133.1 Offset

Register	Offset	Description
ORSR_JR0	102Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
ORSR_JR1	202Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
ORSR_JR2	302Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.133.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved						ORS									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.133.3 Fields

Field	Description
31-10 —	Reserved
9-0 ORS	Output Ring Size. (measured in number of entries)

10.13.134 Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR2)

The Output Ring Jobs Removed Register tells CAAM how many jobs were removed from the output ring for processing by software. Because there are 3 Job Rings, there are 3 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. 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. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section Job Ring Registers.

More information on job management can be found in Job Ring interface.

10.13.134.1 Offset

Register	Offset	Description
ORJRR_JR0	1034h	(used by JR 0) Accessible only when using DID specified in JR0DID register
ORJRR_JR1	2034h	(used by JR 1) Accessible only when using DID specified in JR1DID register
ORJRR_JR2	3034h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.134.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved						ORJR									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.134.3 Fields

Field	Description
31-10 —	Reserved
9-0 ORJR	Output Ring Jobs Removed. (measured in number of entries)

10.13.135 Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR2)

The Output Ring Slots Full Register tells the software how many completed jobs CAAM has placed in the output ring. Because there are 3 Job Rings, there are 3 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 Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR2)). The value in the Output Ring Slots Full Register cannot be larger than the value in the Output Ring Size Register (see Section Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR2)).

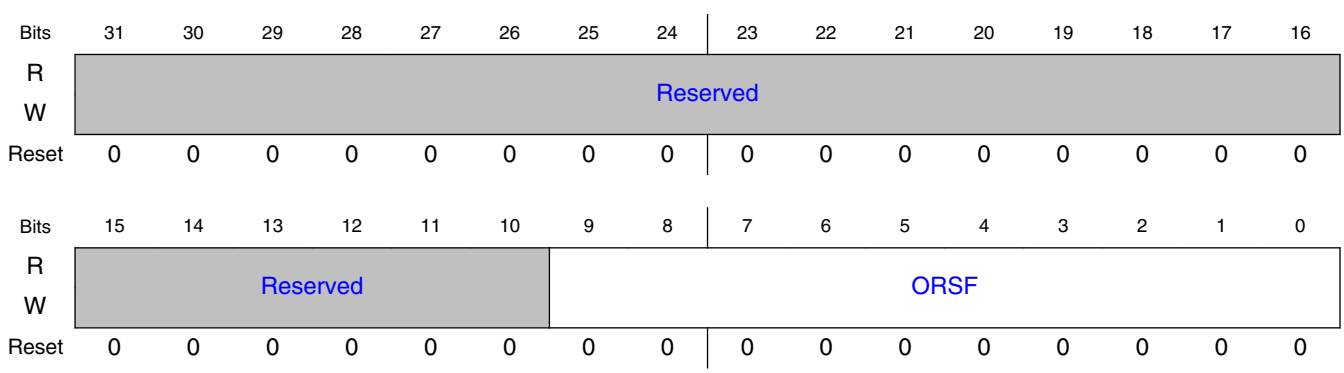
The Job Ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, ORSFR may only be written with a transaction with ns=0. See Section Job Ring Registers.

More information on job management can be found in Job Ring interface.

10.13.135.1 Offset

Register	Offset	Description
ORSFR_JR0	103Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
ORSFR_JR1	203Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
ORSFR_JR2	303Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.135.2 Diagram



10.13.135.3 Fields

Field	Description
31-10 —	Reserved
9-0 ORSF	Output Ring Slots Full. (measured in number of entries)

10.13.136 Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR2)

This register is used to show the status of the last job that was completed. Because there are 3 Job Rings, there are 3 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 0-31 of this register are written into the output ring after the completion of a job, and software should read the status from there. More information on Job Ring management can be found in Section Job Ring interface. Only one type of error will be valid at a time. The status code and various other information related to the status are given in the SSED field.

10.13.136.1 Offset

Register	Offset	Description
JRSTAR_JR0	1044h	(used by JR 0) Accessible only when using DID specified in JR0DID register
JRSTAR_JR1	2044h	(used by JR 1) Accessible only when using DID specified in JR1DID register
JRSTAR_JR2	3044h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.136.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SSRC				SSED											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SSED															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.136.3 Fields

Field	Description
31-28 SSRC	Status source. These bits define which source is reporting the status. All other values - reserved 0000 - No Status Source (No Error or Status Reported) 0001 - Reserved 0010 - CCB Status Source (CCB Error Reported) 0011 - Jump Halt User Status Source (User-Provided Status Reported) 0100 - DECO Status Source (DECO Error Reported) 0101 - Reserved 0110 - Job Ring Status Source (Job Ring Error Reported) 0111 - Jump Halt Condition Codes (Condition Code Status Reported)
27-0 SSED	Source-specific error details. The format of this field depends on the status source specified in the SSRC field. The interpretation of the SSED field for all status sources is shown in Job termination status/error codes.

10.13.137 Job Ring Interrupt Status Register for Job Ring a (JRINTR_JR0 - JRINTR_JR2)

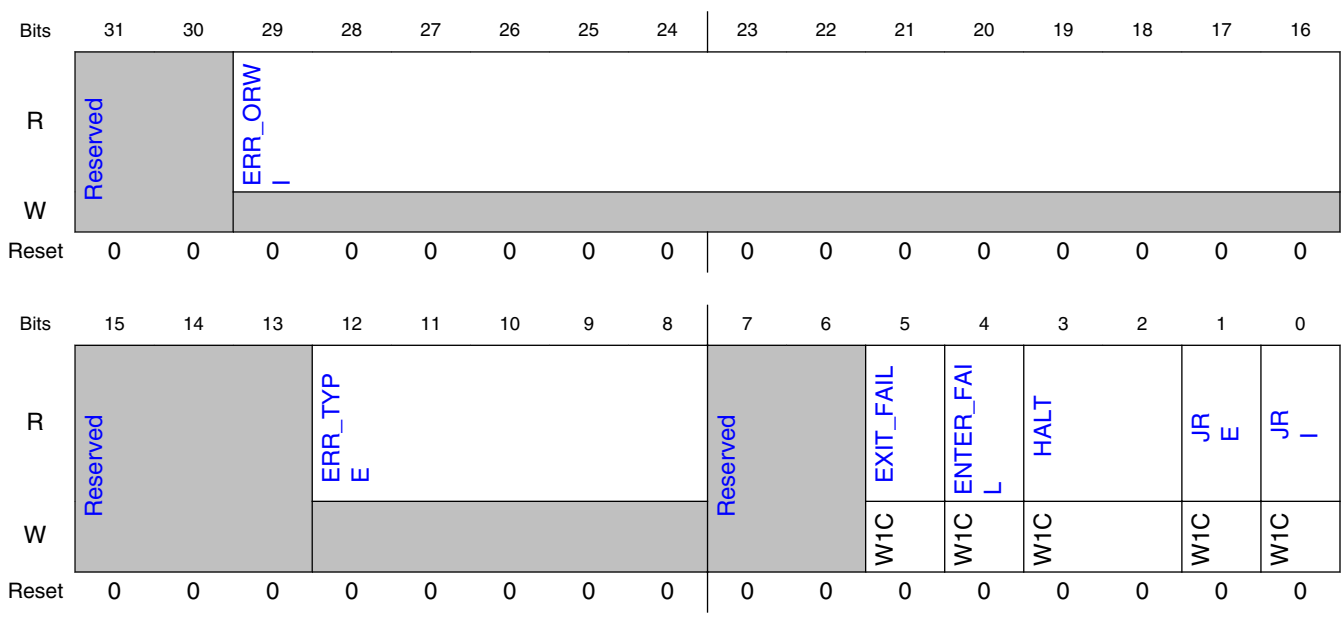
The Job Ring Interrupt Status Register indicates whether CAAM has asserted an interrupt for a particular Job Ring, whether software has requested that the Job Ring be halted, whether the Job Ring is now halted, and whether there is an error in this Job Ring. If there was an error, the type of error is 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 3 Job Rings, there are 3 copies of this register.

10.13.137.1 Offset

Register	Offset	Description
JRINTR_JR0	104Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
JRINTR_JR1	204Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
JRINTR_JR2	304Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.137.2 Diagram



10.13.137.3 Fields

Field	Description
31-30 —	Reserved

Table continues on the next page...

Field	Description
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 Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR2)).
15-13 —	Reserved
12-8 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 Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR2).) 00001 - Error writing status to Output Ring 00011 - Bad input ring base address (not on a 4-byte boundary). 00100 - Bad output ring base address (not on a 4-byte boundary). 00101 - Invalid write to Input Ring Base Address Register or Input Ring Size Register. Can be written when there are no jobs in the input ring or when the Job Ring is halted. [1] 00110 - Invalid write to Output Ring Base Address Register or Output Ring Size 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.[1] 00111 - Job Ring reset released before Job Ring is halted.[1] 01000 - Removed too many jobs (ORJRR larger than ORSFR).[1] 01001 - Added too many jobs (IRJAR larger than IRSAR).[1] 01010 - Writing ORSF > ORS [2] 01011 - Writing IRSA > IRS [2] 01100 - Writing ORWI > ORS in bytes [2] 01101 - Writing IRRI > IRS in bytes [2] 01110 - Writing IRSA when ring is active [2] 01111 - Writing IRRI when ring is active [2] 10000 - Writing ORSF when ring is active [2] 10001 - Writing ORWI when ring is active [2]
7-6 —	Reserved
5 EXIT_FAIL	Exit Fail Mode. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the EXIT_FAIL bit will clear it.
4 ENTER_FAIL	Enter Fail Mode. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the ENTER_FAIL bit will clear it.
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 (JRCR_JRn[RESET]). 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.

Table continues on the next page...

Field	Description
	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 CAAM is in the Fail Mode and the FAIL MODE bit is not set in the Job Ring Configuration Register (JRCFGR_JRn_MS[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 CAAM transitions out of FAIL MODE. If the JRCFGR_JRn_MS[FAIL_MODE] bit is set, the Job Ring is not halted in FAIL MODE.
1 JRE	Job Ring Error. A Job Ring error occurred. The error code is indicated in the ERR_TYPE field in this register. 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.

1. 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.
2. In these error cases the write is ignored, the interrupt is asserted (unless masked) and the error bit and error_type fields are set in the Job Ring Interrupt Status Register.

10.13.138 Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR2_MS)

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 3 Job Rings, there are 3 copies of this register. Since there are more than 32 bits in the JRCFG Register, it is accessed as two 32-bit words.

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 10-245. Control Data vs. Message Data

Control Data	Message Data
Control data read by CAAM DMA: • Descriptors or other data loaded into the Descriptor Buffer • Job Ring input ring entries • Address pointers • Scatter/Gather Tables • Data loaded into the Class 1 or Class 2 Key Size registers	Message data read by CAAM DMA: • Data read into the Input Data FIFO • Data loaded into the Output Data FIFO • Data loaded into the Class 1 or Class 2 Context registers • Data loaded into the Class 1 or Class 2 Key registers • Data loaded into the Input or Output Data FIFO Nibble Shift registers • Data put into the Auxiliary Data FIFO

Table continues on the next page...

Table 10-245. Control Data vs. Message Data (continued)

Control Data	Message Data
• Data loaded into the Class 1 or Class 2 Data Size registers • Data loaded into the Class 1 or Class 2 ICV Size registers • Data loaded into the DECO DID register • Data loaded into the CHA Control register • Data loaded into the DECO Control register • Data loaded into the IRQ Control register • Data loaded into the DECO Protocol Override register • Data loaded into the Clear Written register • Data loaded into the Math registers • Data loaded into the AAD Size register • Data loaded into the Class 1 IV Size register • Data loaded into the Alternate Data Size Class 1 register • Data loaded into the PKHA Size registers • Data loaded into the iNformation FIFO (NFIFO)	
Control data written by CAAM DMA: • Descriptors or other data stored from the Descriptor Buffer • Job Ring output ring entries • Address pointers • Scatter/Gather Tables • Data stored from the Class 1 or Class 2 Mode registers • Data stored from the DECO Job Queue Control register • Data stored from the Class 1 or Class 2 Key Size registers • Data stored from the DECO Descriptor Address Register • Data stored from the Class 1 or Class 2 Data Size registers • Data stored from the DECO Status register • Data stored from the Class 1 or Class 2 ICV Size registers • Data stored from the CHA Control register • Data stored from the IRQ Control register • Data stored from the Clear Written register • Data stored from the Math registers • Data stored from the CCB Status register • Data stored from the AAD Size register • Data stored from the Class 1 IV Size register • Data stored from the PKHA Size registers	Message data written by CAAM DMA: • Data output via the Output Data FIFO • Data stored from the Class 1 or Class 2 Context registers

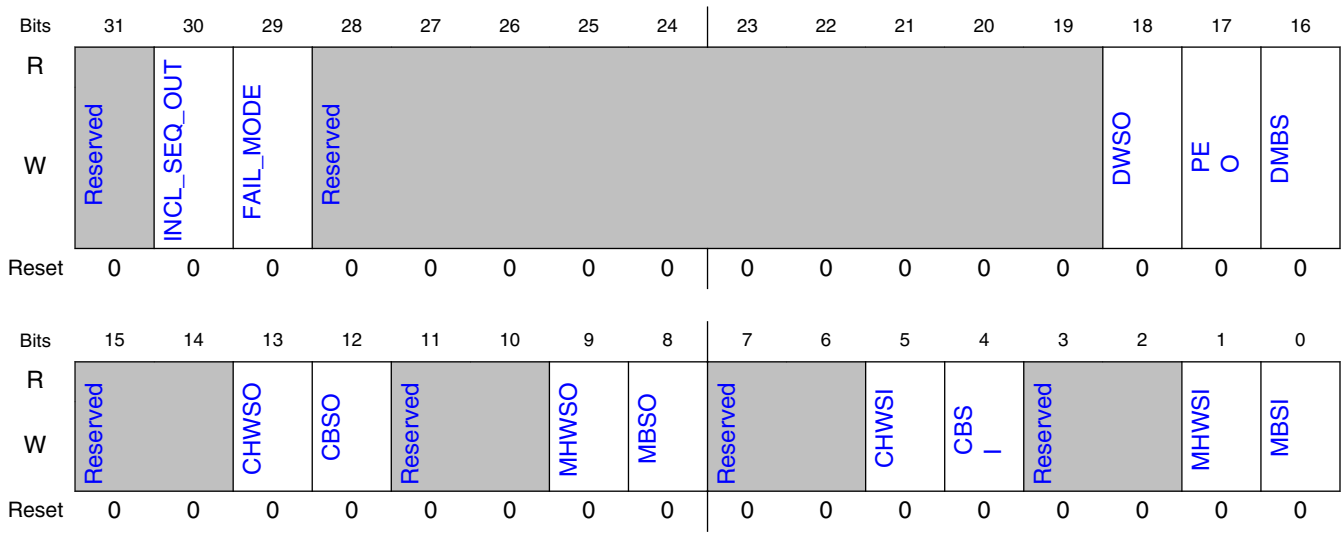
10.13.138.1 Offset

Register	Offset	Description
JRCFGR_JR0_MS	1050h	(used by JR 0) Accessible only when using DID specified in JR0DID register

Table continues on the next page...

Register	Offset	Description
JRCFGR_JR1_MS	2050h	(used by JR 1) Accessible only when using DID specified in JR1DID register
JRCFGR_JR2_MS	3050h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.138.2 Diagram



10.13.138.3 Fields

Field	Description
31 —	Reserved
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 FAIL_MODE	Fail mode control. If this bit is set to 1 and CAAM indicates a FAIL MODE, the Job Ring will assert its interrupt and set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The Job Ring will not halt, but will continue to process any available jobs. DECO will return these jobs with a FAIL MODE error. If CAAM transitions out of FAIL MODE, the Job Ring will assert its interrupt and set the EXIT_FAIL bit in the Job Ring Interrupt Status register. If this bit is set to 0 and CAAM indicates a FAIL MODE, the Job Ring will set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The Job Ring will halt until CAAM transitions out of FAIL MODE. When the Job Ring has halted, it will assert its interrupt. If CAAM transitions out of FAIL MODE, the Job Ring will set the EXIT_FAIL bit in the Job Ring Interrupt Status register.

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>28-19
—</td><td>Reserved</td></tr>
<tr><td>18
DWSO</td><td>Double Word Swap Override. Setting DWSO=1 complements the swap control determined by MCFGR[DWT] and JRCFGR_JR[PEO]</td></tr>
<tr><td>17
PEO</td><td>Platform Endian Override - The bit is XORed with the PLEND bit in the CaCSTA 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.</td></tr>
<tr><td>16
DMBS</td><td>Descriptor Message Data Byte Swap (this applies only to internal message data transfers to/from DECO Descriptor Buffers). An example is shown below:

<table><tr><th>Data as it is read from the source</th><th>0123456789abcdefh</th></tr><tr><td>Data as it is written to the destination when DBMS = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as it is written to the destination when DBMS = 1</td><td>23016745ab89efcdh</td></tr></table></td></tr>
<tr><td>15-14
—</td><td>Reserved</td></tr>
<tr><td>13
CHWSO</td><td>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><tr><th>Data as interpreted within CAAM</th><th>0123456789abcdefh</th></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR CHWSO = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR CHWSO = 1</td><td>45670123cdef89abh</td></tr></table></td></tr>
<tr><td>12
CBSO</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of control data written by CAAM DMA. An example is shown below:

<table><tr><th>Data as interpreted within CAAM</th><th>0123456789abcdefh</th></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR CBSO = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR CBSO = 1</td><td>23016745ab89efcdh</td></tr></table></td></tr>
<tr><td>11-10
—</td><td>Reserved</td></tr>
<tr><td>9
MHWSO</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of message data written by CAAM DMA. An example is shown below:

<table><tr><th>Data as interpreted within CAAM</th><th>0123456789abcdefh</th></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR MHWSO = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR MHWSO = 1</td><td>45670123cdef89abh</td></tr></table></td></tr>
</table>

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8
MBSO</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of message data written by CAAM DMA. An example is shown below:

<table><tr><th>Data as interpreted within CAAM</th><th>0123456789abcdefh</th></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR MBSO = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as written to memory when PEO XOR PLEND XOR MBSO = 1</td><td>23016745ab89efcdh</td></tr></table></td></tr>
<tr><td>7-6
—</td><td>Reserved</td></tr>
<tr><td>5
CHWSI</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of control data read by CAAM DMA. An example is shown below:

<table><tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR CHWSI = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR CHWSI = 1</td><td>45670123cdef89abh</td></tr></table></td></tr>
<tr><td>4
CBSI</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of control data read by CAAM DMA. An example is shown below:

<table><tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR CBSI = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR CBSI = 1</td><td>23016745ab89efcdh</td></tr></table></td></tr>
<tr><td>3-2
—</td><td>Reserved</td></tr>
<tr><td>1
MHWSI</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of message data read by CAAM DMA. An example is shown below:

<table><tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR MHWSI = 0</td><td>0123456789abcdefh</td></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR MHWSI = 1</td><td>45670123cdef89abh</td></tr></table></td></tr>
<tr><td>0
MBSI</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of message data read by CAAM DMA. An example is shown below:

<table><tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr><tr><td>Data as interpreted by CAAM when PEO XOR PLEND XOR MBSI = 0</td><td>01234567ababcdefh</td></tr></table></td></tr>
</table>

Field	Description
	Data as stored in memory | **0123456789abcdefh** Data as interpreted by CAAM when PEO XOR PLEND XOR MBSI = 1 | 23016745ab89efcdh

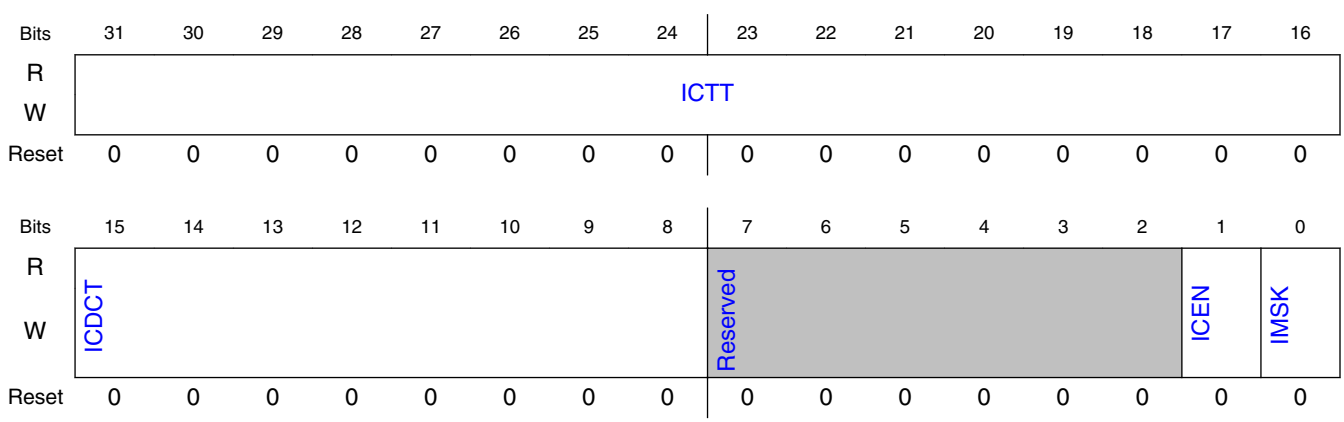
10.13.139 Job Ring Configuration Register for Job Ring a, least-significant half (JRCFGR_JR0_LS - JRCFGR_JR2_LS)

See description of JRCFGR_JR_MS

10.13.139.1 Offset

Register	Offset	Description
JRCFGR_JR0_LS	1054h	(used by JR 0) Accessible only when using DID specified in JR0DID register
JRCFGR_JR1_LS	2054h	(used by JR 1) Accessible only when using DID specified in JR1DID register
JRCFGR_JR2_LS	3054h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.139.2 Diagram



10.13.139.3 Fields

Field	Description
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.
7-2 —	Reserved
1 ICEN	Interrupt Coalescing Enable. 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 > 0), then the interrupt will clear but reassert on the next clock cycle. 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 >= ICDCT), then the interrupt will clear but reassert on the next clock cycle.
0 IMSK	Interrupt Mask. Mask the interrupt that is associated with the particular processor. 0 - Interrupt enabled. 1 - Interrupt masked.

10.13.140 Input Ring Read Index Register for Job Ring a (IRRIR_JR0 - IRRIR_JR2)

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 3 Job Rings, there are 3 copies of this register.

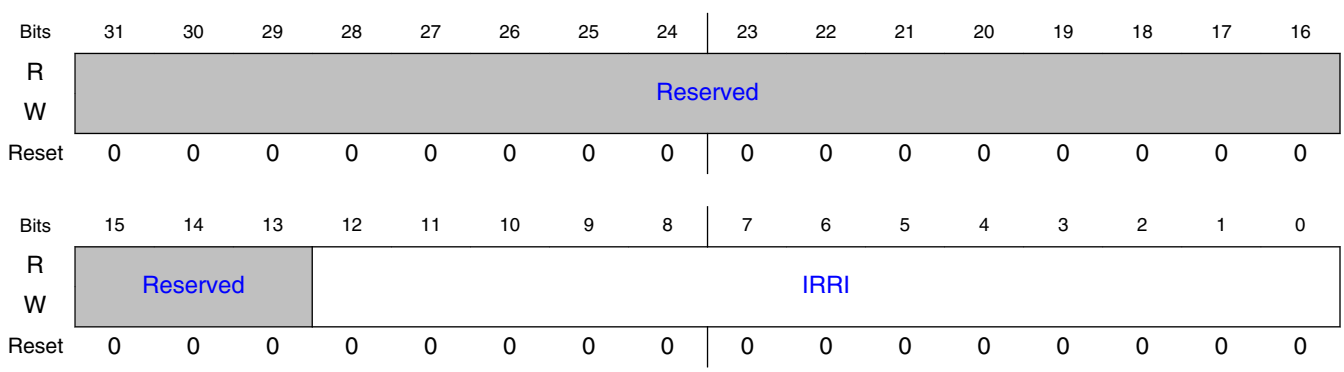
The Job Ring must be started in order to write the IRRIR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, IRRIR may only be written with a transaction with ns=0.

More information on job management and job ring registers can be found in chapters Job Ring interface and Job Ring Registers.

10.13.140.1 Offset

Register	Offset	Description
IRRIR_JR0	105Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
IRRIR_JR1	205Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
IRRIR_JR2	305Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.140.2 Diagram



10.13.140.3 Fields

Field	Description
31-13	Reserved

Table continues on the next page...

Field	Description
—	
12-0 IRRI	Input Ring Read Index.

10.13.141 Output Ring Write Index Register for Job Ring a (ORWIR_JR0 - ORWIR_JR2)

The Output Ring Write Index Register points to the tail of the queue within the output ring. Because there are 3 Job Rings, there are 3 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 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 if INCL_SEQ_OUT=0 the increment will be 8. If INCL_SEQ_OUT=1, the increment will be 12. For a discussion of address pointers see Address pointers.

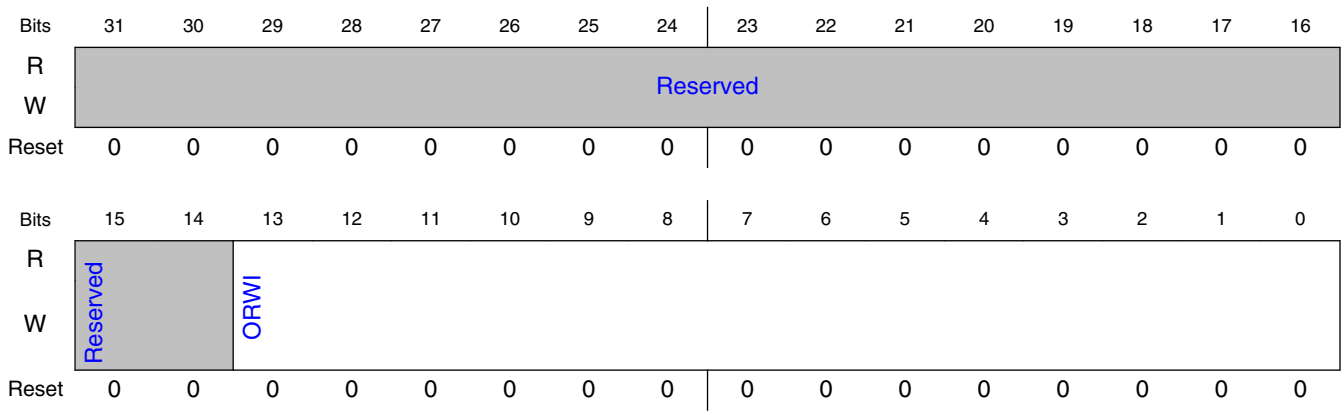
The Job Ring must be started in order to write the ORWIR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, ORWIR may only be written with a transaction with ns=0.

More information on job management and job ring registers can be found in chapters Job Ring interface and Job Ring Registers.

10.13.141.1 Offset

Register	Offset	Description
ORWIR_JR0	1064h	(used by JR 0) Accessible only when using DID specified in JR0DID register
ORWIR_JR1	2064h	(used by JR 1) Accessible only when using DID specified in JR1DID register
ORWIR_JR2	3064h	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.141.2 Diagram



10.13.141.3 Fields

Field	Description
31-14 —	Reserved
13-0 ORWI	Output Ring Write Index. The pointer to the next entry in the output ring.

10.13.142 Job Ring Command Register for Job Ring a (JRCR_JR0 - JRCR_JR2)

Software can use this register to issue a park, 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 tanks or DECOs. 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. When JRCR[RESET] is written with a 1, the HALT field in the Job Ring Interrupt Status register will display 01b. When the flush operation is complete, the HALT field will change to 10b. After a flush completes, the halt can be cleared and job processing will resume, or a reset can be requested.

A park command will stall any jobs in the job ring that have not yet been fetched, but will allow all the jobs in progress to complete normally. A park command may be issued only if virtualization is enabled. If virtualization is disabled, any writes to the PARK bit are ignored.

During the time between the write to PARK and all of the in-progress jobs completing, the HALT field in the Job Ring Interrupt Status register will return 01b indicating that the Job Ring was asked to stop processing jobs. When all jobs are complete and the Job Ring has halted, the Job Ring Interrupt Status register will indicate this by setting the HALT field to 10b. Once the Job Ring indicates that it has halted, it is safe to read the values of the Job Ring registers to save the Job Ring state. The following register values should be saved:

- JRCFGR_JR -Job Ring Configuration Register for the Job Ring
- IRBAR_JR - Input Ring Base Address Register for the Job Ring
- IRSR_JR - Input Ring Size Register for the Job Ring
- ORBAR_JR - Output Ring Base Address Register for the Job Ring
- ORSR_JR - Output Ring Size Register for the Job Ring
- IRSAR_JR - Input Ring Slots Available Register for the Job Ring
- ORSFR_JR - Output Ring Slots Full Register for the Job Ring
- IRRIR_JR - Input Ring Read Index Register for the Job Ring
- ORWIR_JR - Output Ring Write Index Register for the Job Ring

Once the state is saved, the Job Ring may be reassigned. To reassign the Job Ring, the registers that were saved should be rewritten with new values. When reprogramming, note that IRS must be written before IRSA or IRRI, and ORS must be written before ORSF or ORWI. IRSA should be written last because this is the register that indicates to the Job Ring that it has jobs to process. Failure to write the registers in the correct order may result in one of the following errors: IRSA>IRS, IRRI>IRS, ORSF>ORS, or ORWI>ORS. Once the Job Ring is reprogrammed, park status can be released so that the Job Ring can start running again. To do this, write a "1" to the MSB of the HALT field in the Job Ring Interrupt Status register. Note that if software tries to release parking status before the Job Ring has halted, a fatal error will occur (type 00111). This is the same error type as releasing the Job Ring from reset status before the ring has halted.

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.

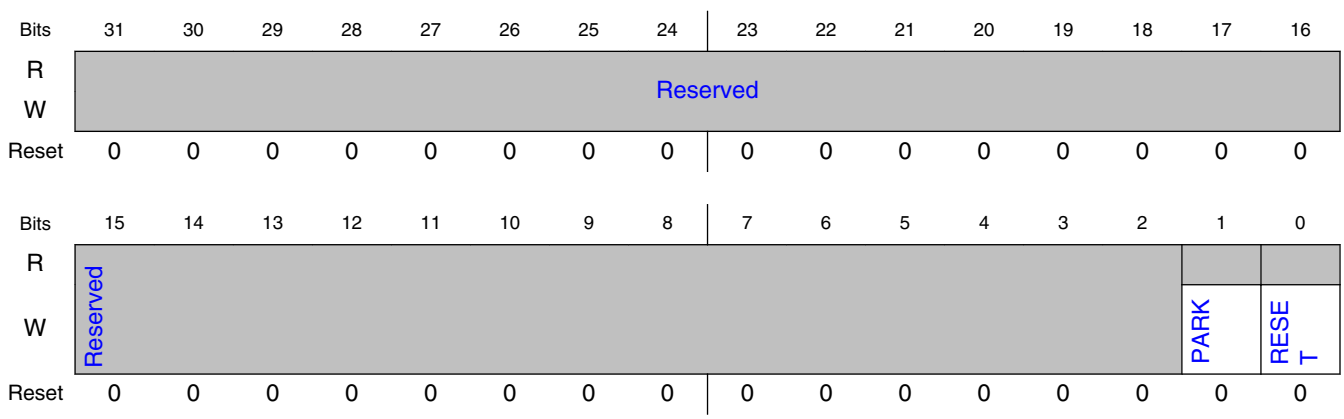
A reset can be initiated only after a flush has been requested and completed as indicated by the HALT field in the Job Ring Interrupt Status Register. After a reset, job processing will resume when the Input Ring Jobs Added Register is written to indicate that new jobs are available. If both PARK and RESET are written to 1, either a flush or a reset will be performed depending on the current value of RESET.

Because there are 3 Job Rings, there are 3 copies of this register.

10.13.142.1 Offset

Register	Offset	Description
JRCR_JR0	106Ch	(used by JR 0) Accessible only when using DID specified in JR0DID register
JRCR_JR1	206Ch	(used by JR 1) Accessible only when using DID specified in JR1DID register
JRCR_JR2	306Ch	(used by JR 2) Accessible only when using DID specified in JR2DID register

10.13.142.2 Diagram



10.13.142.3 Fields

Field	Description
31-2 —	Reserved

Table continues on the next page...

Field	Description
1 PARK	Park - When PARK is 0, software writes a 1 to PARK to request that the Job Ring be "parked", i.e. quiesced. All jobs currently "in flight" (in holding tanks, DECOs or waiting for status results to be written out) are completed, but no new jobs are fetched from the input ring. When the job ring has completed parking, the HALT field in the Job Ring Interrupt Status register will change from 01b to 10b.
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.

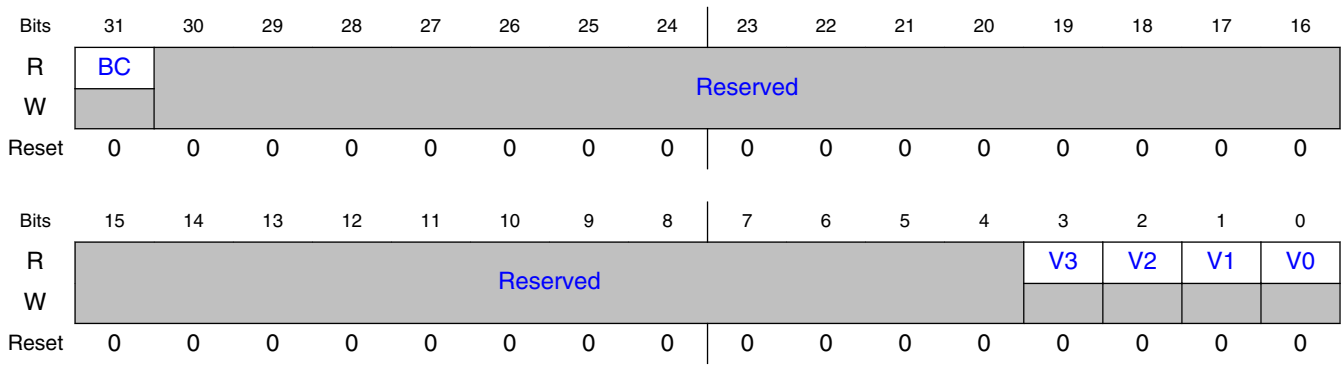
10.13.143 Job Ring a Address-Array Valid Register (JR0AAV - JR2AAV)

The Job Ring Address-Array Valid register indicates which entries stored in the Job Ring Address-Array Address Registers contain valid data. The register is intended to be used when debugging descriptor execution via a Job Ring. The Debug Control Register can be used to stop CAAM processing before reading the Job Ring debug registers so that a consistent set of values can be read. Note that this version of CAAM implements 4 Job Ring Address-Array Registers.

10.13.143.1 Offset

Register	Offset	Description
JR0AAV	1704h	Used with Job Ring 0
JR1AAV	2704h	Used with Job Ring 1
JR2AAV	3704h	Used with Job Ring 2

10.13.143.2 Diagram



10.13.143.3 Fields

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 JR0AAA0 is read, and BC is then set to 1 if the content of any of the JRAAAx registers or the JRaAAV register changes (due to new addresses being loaded into AA or existing addresses being sent to a holding tank) before the JRaAAV 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.
30-4 —	Reserved
3 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.
2 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.
1 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.
0 V0	Valid 0. When V0=1, 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.

10.13.144 Job Ring a Address-Array Address b Register (JR0AAA0 - JR2AAA3)

JRaAAAb registers are used when debugging descriptor execution started from a Job Ring. The Debug Control Register (DEBUGCTL) can be used to stop CAAM processing before reading Job Ring debug registers so that a consistent set of values can be read.As discussed in Job scheduling, the job Queue Controller may buffer up to 4 Job Descriptor addresses for each Job Ring and read up to 4 Job Descriptor addresses from a Job Ring before servicing the next Job Ring in round-robin fashion. For optimal performance (primarily to minimize bus transactions), CAAM will read up to 4 job descriptor addresses in a single bus burst when possible. JRaAAAb registers are used to buffer the job descriptor addresses after the job queue controller fetches the descriptor address from the input ring and before assigning the descriptor to a Holding Tank.

10.13.144.1 Offset

For a = 0 to 2; b = 0 to 3:

Register	Offset	Description
JRaAAAb	1800h + (a × 1000h) + (b × 8h)	Used with Job Ring a. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.144.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JD_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.144.3 Fields

Field	Description
63-32 —	Reserved
31-0 JD_ADDR	Job Descriptor Address.

10.13.145 Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR2)

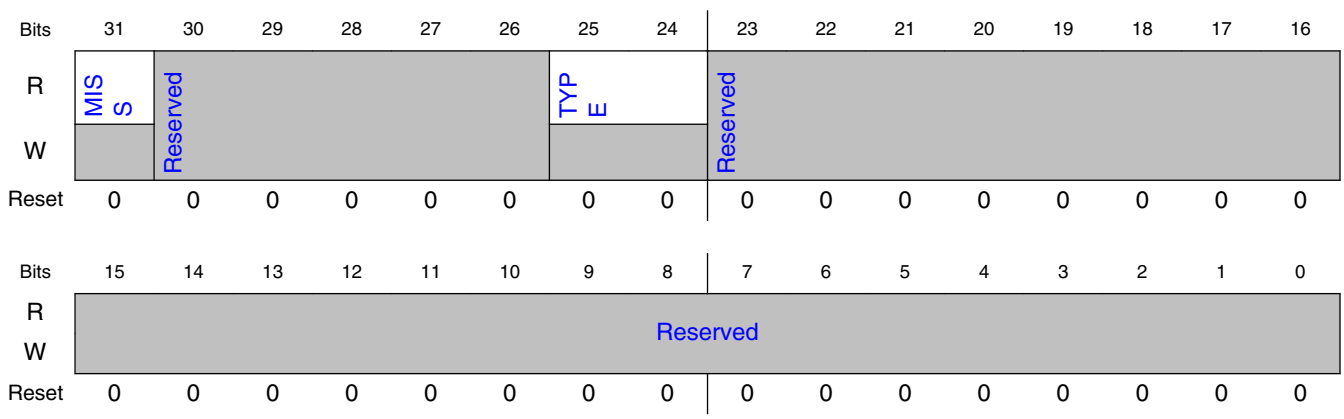
If a recoverable error occurs related to execution of a job from a Job Ring, error information will be captured in the JR's REIR registers. Data for a second recoverable error related to jobs from JR will not be captured until the REIR0JR is written. If another

bus error from JR occurs before then, the double error status bit (MISS) in REIR0JR will be set. When REIR0JR is written, all of JR's REIRJR registers are cleared and error capture is re-enabled.

10.13.145.1 Offset

Register	Offset	Description
REIR0JR0	1E00h	(used by JR 0)
REIR0JR1	2E00h	(used by JR 1)
REIR0JR2	3E00h	(used by JR 2)

10.13.145.2 Diagram



10.13.145.3 Fields

Field	Description
31 MISS	If MISS=1, a second recoverable error associated with JR occurred before REIR0JR was written following a previous JR recoverable error.
30-26 —	Reserved
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 0 : reserved If TYPE = 1 : memory access error If TYPE = 2 : reserved If TYPE = 3 : reserved

Table continues on the next page...

Field	Description
23-0 —	Reserved

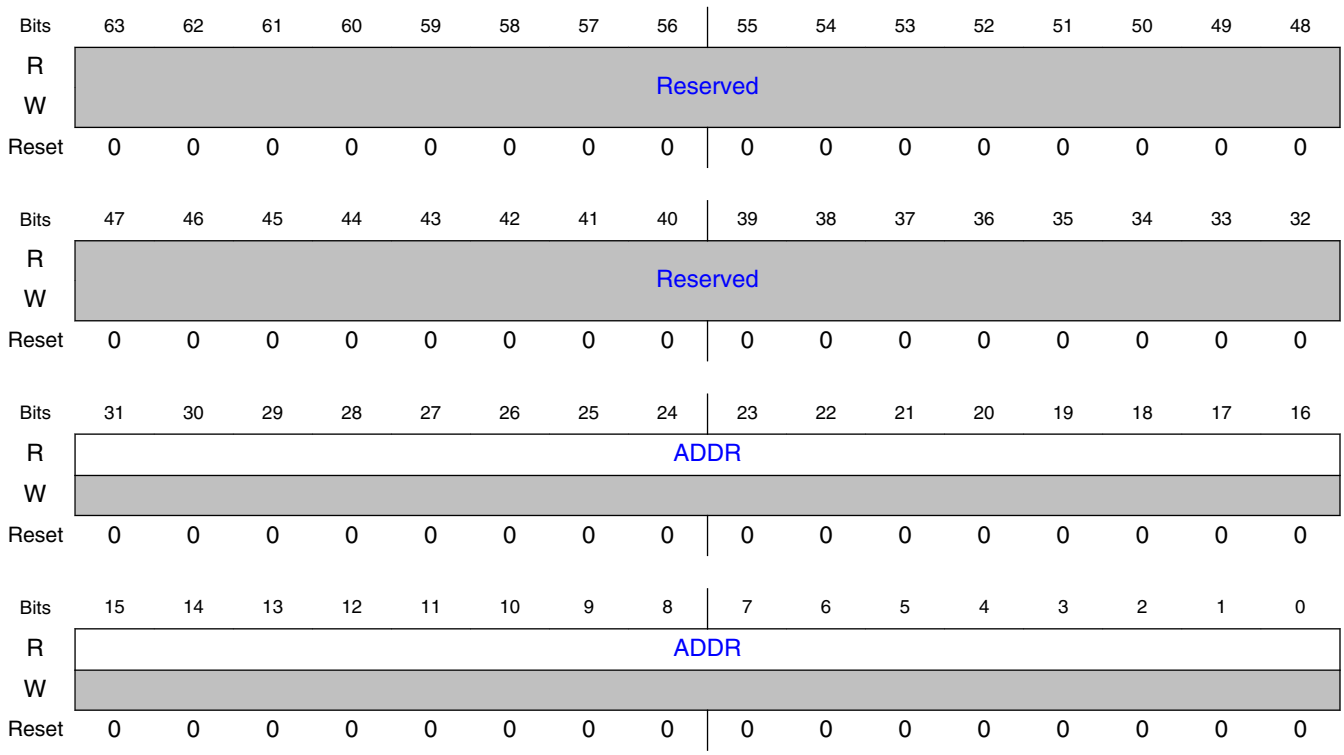
10.13.146 Recoverable Error Indication Record 2 for Job Ring a (REIR2JR0 - REIR2JR2)

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR2).

10.13.146.1 Offset

Register	Offset	Description
REIR2JR0	1E08h	Used by JR0. Accessible only when using DID specified in JR0DID register
REIR2JR1	2E08h	Used by JR1. Accessible only when using DID specified in JR1DID register
REIR2JR2	3E08h	Used by JR2. Accessible only when using DID specified in JR2DID register

10.13.146.2 Diagram



10.13.146.3 Fields

Field	Description
63-32 —	Reserved
31-0 ADDR	Address associated with the recoverable JR error.

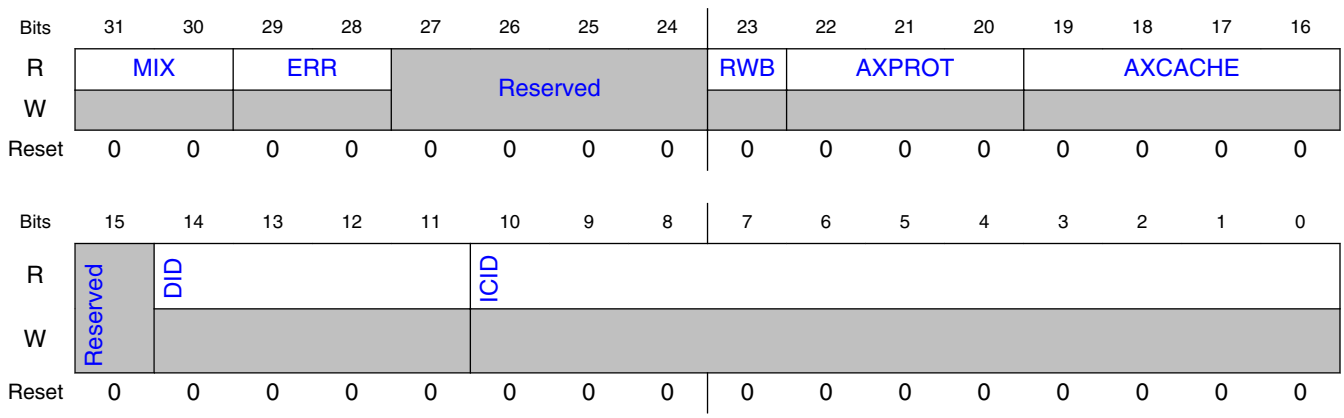
10.13.147 Recoverable Error Indication Record 4 for Job Ring a (REIR4JR0 - REIR4JR2)

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR2).

10.13.147.1 Offset

Register	Offset	Description
REIR4JR0	1E10h	(used by JR 0)
REIR4JR1	2E10h	(used by JR 1)
REIR4JR2	3E10h	(used by JR 2)

10.13.147.2 Diagram



10.13.147.3 Fields

Field	Description
31-30 MIX	This field holds the memory interface index associated with the recoverable error.
29-28 ERR	This field holds the AXI error response associated with the recoverable error.
27-24 —	Reserved
23 RWB	This field specifies whether the memory access was a read or write.
22-20 AXPROT	This field holds the AXI protection transaction attribute used for the memory access.
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15	Reserved

Table continues on the next page...

Field	Description
—	
14-11 DID	This field holds the DID associated with the recoverable error.
10-0 ICID	This field holds the ICID associated with the recoverable error.

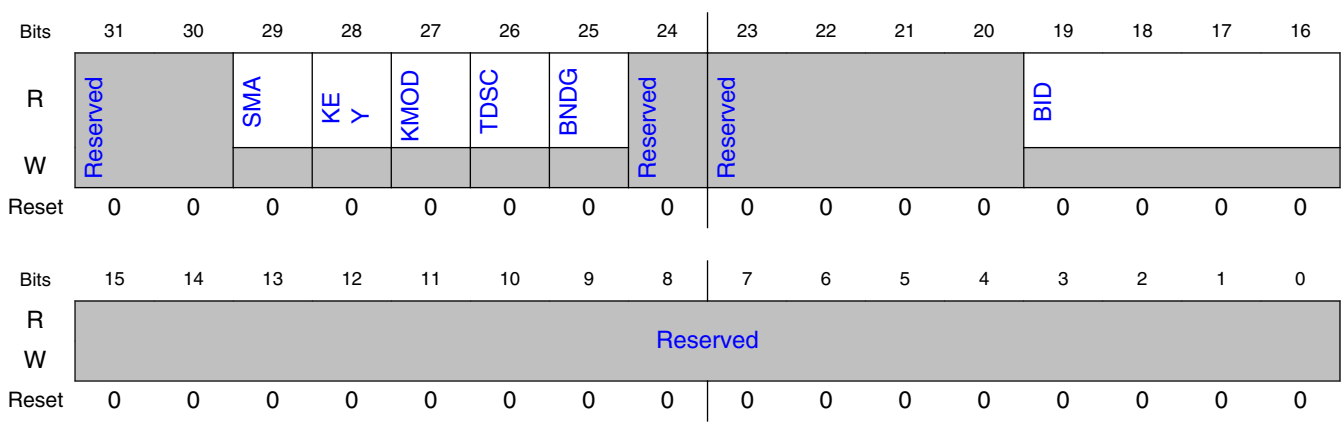
10.13.148 Recoverable Error Indication Record 5 for Job Ring a (REIR5JR0 - REIR5JR2)

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR2).

10.13.148.1 Offset

Register	Offset	Description
REIR5JR0	1E14h	(used by JR 0)
REIR5JR1	2E14h	(used by JR 1)
REIR5JR2	3E14h	(used by JR 2)

10.13.148.2 Diagram



10.13.148.3 Fields

Field	Description
31-30 —	Reserved
29 SMA	This field indicates whether the bus transaction associated with the recoverable error was an attempted access to CAAM Secure Memory.
28 KEY	This field indicates whether the bus transaction associated with the recoverable error was an attempted key access to CAAM Secure Memory.
27 KMOD	This field indicates whether the bus transaction associated with the recoverable error was an attempt to read the key modifier associated with an CAAM Secure Memory partition.
26 TDSC	This field indicates whether the bus transaction associated with the recoverable error was an attempt to assert trusted descriptor privilege when accessing an CAAM Secure Memory partition.
25 BNDG	This field indicates whether the bus transaction associated with the recoverable error was initiating an CAAM Secure Memory blob operation.
24 —	Reserved
23-20 —	Reserved
19-16 BID	This field holds the block identifier (see Table 10-243) of the source of the AXI transaction associated with the recoverable error.
15-0 —	Reserved

10.13.149 RTIC Status Register (RSTA)

This section describes the registers of the Run Time Integrity Checker (RTIC). A functional description of the RTIC can be found in Run-time Integrity Checker (RTIC). Note the use of the RTIC is optional, to support platform assurance.

The Run Time Integrity Checking Status Register is a read-only register that gives software information about the internal states of RTIC. Reading the RTIC Status Register will clear all errors and the RTIC interrupt. Due to timing issues, instead of polling this register software should read the RTIC Status Register after an RTIC done interrupt.

10.13.149.1 Offset

Register	Offset
RSTA	6004h

10.13.149.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved					CS			Reserved				RTD	HOD	ABH	WE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				AE				MIS				HE	SV	HD	BSY
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.149.3 Fields

Field	Description
31-27 —	Reserved
26-25 CS	RTIC Current State. Indicates the current state of the RTIC controller. 00 - Idle State 01 - Single Hash State 10 - Run-time State 11 - Error State
24-20 —	Reserved
19 RTD	Run Time Blocks Disabled. When RTIC is in Run Time mode, a 1 in the RTD field indicates that all the Memory Blocks are Disabled for Run Time Operation.
18 HOD	Hash Once Blocks Disabled. All the Memory Blocks are Disabled for Hash Once Operation. This bit is set when RTIC is either in an Idle State or Hash Once State and none of the Memory Blocks have been enabled for Hash Once Operation.
17 ABH	All Blocks Hashed. This is a bit that is used for debugging. This bit toggles during run-time mode every time RTIC completes hashing A-D memory blocks and starts over at the beginning again.

Table continues on the next page...

Field	Description
16 WE	RTIC Watchdog Error. RTIC Watchdog timer has tripped during run-time hashing. This indicates that all enabled memory segments did not finish a round of hashing prior to the RTIC watchdog timer completing. 0 - No RTIC Watchdog timer error has occurred. 1 - RTIC Watchdog timer has expired prior to completing a round of hashing.
15-12 —	Reserved
11-8 AE	Address Error. Indicates an illegal address was read from a peripheral memory block. This is caused by an invalid start address in the Address 1/2 fields or a value in the Length 1/2 fields that caused the RTIC to read outside a peripheral memory's valid address space. If an address error occurs, the illegal address will be captured in the CAAM Fault Address Register (Section Fault Address Register (FAR)). Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error: xxx1 - Memory Block A Address Error xx1x - Memory Block B Address Error x1xx - Memory Block C Address Error 1xxx - Memory Block D Address Error The settings for each bit are as follows: 0000 - All reads by RTIC were valid. 0001 - An illegal address was accessed by the RTIC
7-4 MIS	Memory Integrity Status. Indicates memory block(s) with error. Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error: xxx1 - Memory Block A Hash Error xx1x - Memory Block B Hash Error x1xx - Memory Block C Hash Error 1xxx - Memory Block D Hash Error The settings for each bit are as follows: 0000 - Memory Block X is valid or state unknown 0001 - Memory Block X has been corrupted
3 HE	Hashing Error. Indicates that a unlocked memory block has been corrupted during run time or that an address error has occurred. The unlocked memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. The security violation signal will be asserted. RTIC will generate a done interrupt and disable checking the memory block that caused the failure. Unlocked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)). 0 - Memory block contents authenticated. 1 - Memory block hash doesn't match reference value.
2 SV	Security Violation. Indicates that a locked RTIC memory block has been corrupted during run-time, an address error has occurred, or an RTIC Watchdog timeout has occurred. The memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. If an RTIC Watchdog timeout error occurred then the WE bit will be set. A security violation can only be cleared by a hardware reset.

Table continues on the next page...

Field	Description
	Locked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)). 0 - Memory block contents authenticated. 1 - Memory block hash doesn't match reference value.
1 HD	Hash Once Operation Completed (Hash Done). Processor may read hash values. If an error occurs during hashing or no memory blocks are enabled for one-time hash, this bit will not be set even if the RTIC hardware interrupts are asserted. This bit is cleared by setting the CINT bit in the RTIC Command Register (see Section RTIC Command Register (RCMD)) or when the RTIC enters the run-time checking state. 0 - Boot authentication disabled 1 - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode.
0 BSY	RTIC Idle/Busy Status. When busy, the RTIC cannot be written to. 0 - RTIC Idle. 1 - RTIC Busy.

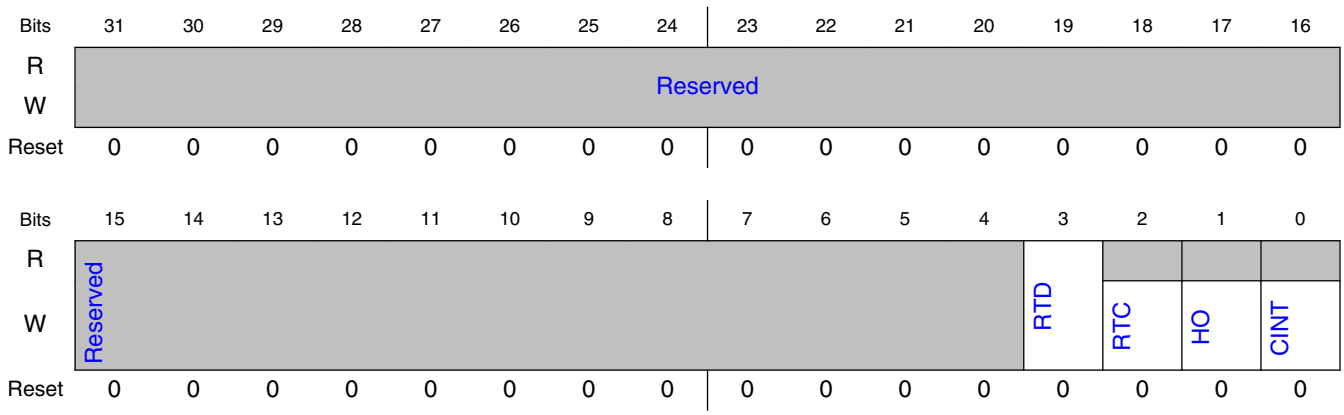
10.13.150 RTIC Command Register (RCMD)

The Run Time Integrity Checking Command Register is used to issue commands to the RTIC. This register is used to instruct the RTIC to perform different functions. This register is only writeable when RTIC is in an idle state.

10.13.150.1 Offset

Register	Offset
RCMD	600Ch

10.13.150.2 Diagram



10.13.150.3 Fields

Field	Description
31-4 —	Reserved
3 RTD	Run Time Disable. Does not allow RTIC to be put into Run-Time mode. This bit will have higher priority in the case where both Run Time Check and Run Time Disable are set on the same write. Run Time Disable is ignored if RTIC is already in the Run Time Mode. 0 - Allow Run Time Mode 1 - Prevent Run Time Mode
2 RTC	Run time check. Starts run-time integrity checking for any blocks having the corresponding RTME bit =1 (see RTIC Status Register (RSTA)). Some of the RTIC registers become read-only. This bit is self-clearing and always returns a logic-0 when read. Setting this bit will clear the ipi_done_int hardware interrupt as well as the HASH DONE bit in the RTIC Status Register. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode. If no memory blocks are enabled, setting the RUN TIME CHK bit will cause the RTIC to enter an idle state while waiting for a memory segment to be enabled. Some registers will be read only. No data is hashed and no interrupts or errors will be generated. 0 - Run-time checking disabled 1 - Verify run-time memory blocks continually
1 HO	Hash once. Starts one-time hash/boot code authentication for any blocks having the corresponding HOME bit =1 (see RTIC Status Register (RSTA)). The resulting hash value is stored in the Hash Register File. This bit is self-clearing and always returns a logic-0 when read. If no memory blocks are enabled, a done interrupt will be immediately generated. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode. 0 - Boot authentication disabled

Table continues on the next page...

Field	Description
	1 - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode.
0 CINT	Clear Interrupt. Clears RTIC hardware interrupt signal. This bit is self-clearing and always returns a logic 0 when read. 0 - Do not clear interrupt 1 - Clear interrupt. This bit cannot be modified during run-time checking mode

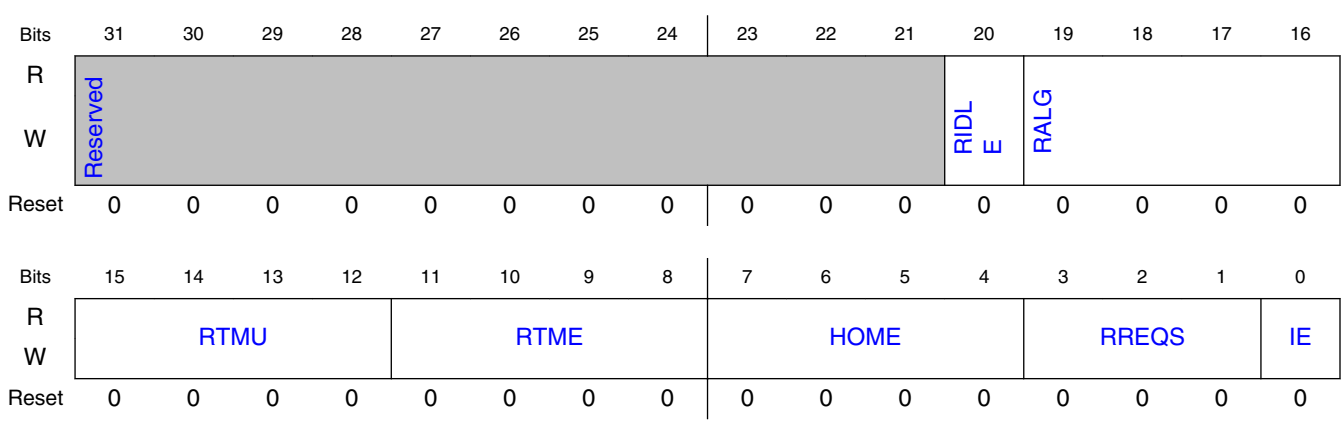
10.13.151 RTIC Control Register (RCTL)

The RTIC is configured by writing to the Run Time Integrity Checking Control Register. No bits in this register are writable unless RTIC is idle or, if RTIC is in Run-Time Mode, unless the control bits for the memory block are disabled and unlocked.

10.13.151.1 Offset

Register	Offset
RCTL	6014h

10.13.151.2 Diagram



10.13.151.3 Fields

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31-21
—</td><td>Reserved</td></tr>
<tr><td>20
RIDLE</td><td>RTIC setting for the IPG_IDLE signal. If RIDLE=1, the signal ipg_idle will be negated if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits is 1). If RIDLE=0 and CAAM is otherwise idle, the signal ipg_idle will still occasionally negate while RTIC is actually hashing a chunk of memory.</td></tr>
<tr><td>19-16
RALG</td><td>RTIC Algorithm Select. Selects which algorithms should be used per Memory Block. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)
<table><tr><td>xxx0b - SHA-256 selected for Memory Block A</td><td>xxx1b - SHA-512 selected for Memory Block A</td></tr><tr><td>xx0xb - SHA-256 selected for Memory Block B</td><td>xx1xb - SHA-512 selected for Memory Block B</td></tr><tr><td>x0xxb - SHA-256 selected for Memory Block C</td><td>x1xxb - SHA-512 selected for Memory Block C</td></tr><tr><td>0xxxb - SHA-256 selected for Memory Block D</td><td>1xxxb - SHA-512 selected for Memory Block D</td></tr></table></td></tr>
<tr><td>15-12
RTMU</td><td>Run Time Memory Unlock. Unlocks memory block(s) for run-time hashing. If a memory block is unlocked it can be enabled and disabled at any time even if RTIC Run-Time Mode has started. These bits are not writable once RTIC Run-Time Mode has started. These bits are intended to allow some of the RTIC memory blocks to be used during RTIC Run-Time Mode by trusted software to verify the integrity of dynamically loaded software. The remaining (locked) memory blocks would be used to verify the integrity of the operating system and the trusted software itself.
xxx1b - Unlock Memory Block A
xx1xb - Unlock Memory Block B
x1xxb - Unlock Memory Block C
1xxxb - Unlock Memory Block D</td></tr>
<tr><td>11-8
RTME</td><td>Run Time Memory Enable. Enables memory block(s) for run-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)
xxx1 - Enable Memory Block A
xx1x - Enable Memory Block B
x1xx - Enable Memory Block C
1xxx - Enable Memory Block D
NOTE: Enable one or more RTIC Memory Blocks for Run Time (e.g. set at least one RTME bit to 1) before changing from Hash Once mode to Run Time mode. Failing to set at least one bit may cause RTIC to hang, which will result in a watchdog timeout.</td></tr>
<tr><td>7-4
HOME</td><td>Hash Once Memory Enable. Enables memory block(s) for one-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)
xxx1 - Enable Memory Block A
xx1x - Enable Memory Block B
x1xx - Enable Memory Block C
1xxx - Enable Memory Block D</td></tr>
</table>

Table continues on the next page...

Field	Description
3-1 RREQS	RTIC Request Size. These bits are used during run-time mode to specify how many blocks of data are hashed every time the throttle counter expires. The block size is determined by the selected Algorithm. Block Size: SHA-256 = 64 bytes SHA-512 = 128 bytes **NOTE:** The RREQS default value of 0 is interpreted as a request for 1 block.
0 IE	Interrupt Enable. Enables the RTIC interrupt. This bit is writable only while RTIC is in an idle state. Hardware interrupts are disabled by default after reset. 0 - Interrupts disabled 1 - Interrupts enabled

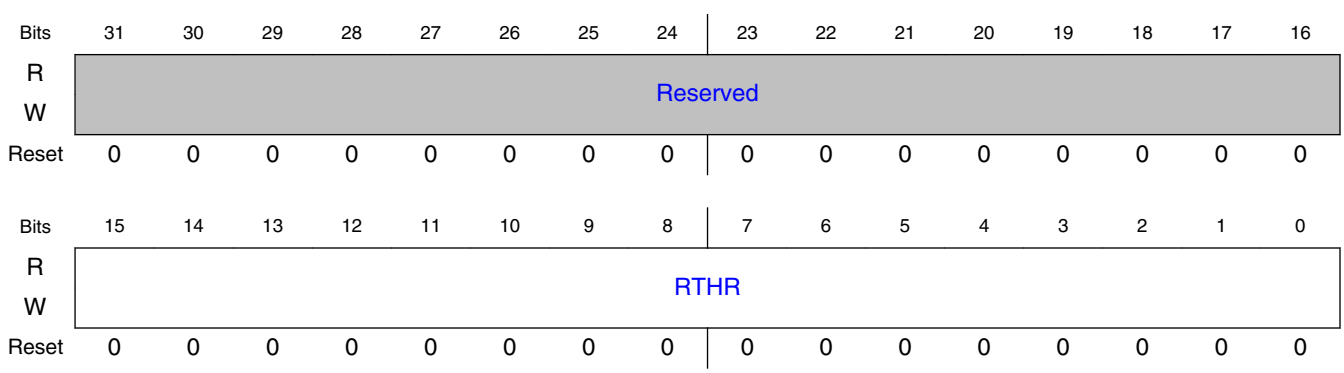
10.13.152 RTIC Throttle Register (RTHR)

The Run Time Integrity Checking Throttle Register can be set to specify how many clock cycles to wait between RTIC hashing operations when RTIC is in run-time mode. The register becomes read-only when RTIC is in run-time mode.

10.13.152.1 Offset

Register	Offset
RTHR	601Ch

10.13.152.2 Diagram



10.13.152.3 Fields

Field	Description
31-16 —	Reserved
15-0 RTHR	Run Time Mode DMA Throttle. Programmable Timer that can be set to specify how many cycles of the 32 Khz clock to wait between RTIC hashing operations during run time mode. At boot time, this register would generally be set to a value that will allow all four memory blocks to be hashed in a reasonable time without high bus utilization.

10.13.153 RTIC Watchdog Timer (RWDOG)

The RTIC Watchdog Register holds the starting value for the RTIC Watchdog Timer, which is used during Run Time Mode to prevent a denial of service attack on RTIC. When RTIC is in Run Time Mode, the RTIC Watchdog Timer begins counting down when run time hashing starts on the first memory block and it resets to the starting value when the last memory block has been hashed. If the RTIC Watchdog Timer times out prior to the last memory block's completion then an RTIC Watchdog error will be generated. Note that the RTIC Watchdog Register is not writable after RTIC enters Run Time Mode, so prior to placing RTIC into Run Time Mode software must write a large enough value into the register to prevent the RTIC Watchdog Timer from expiring under normal conditions. Upon entering low-power mode the RTIC Watchdog Timer will stop counting until low-power mode is exited. Upon exiting low-power mode, the RTIC Watchdog Timer will resume from where it left off.

10.13.153.1 Offset

Register	Offset	Description
RWDOG	6028h	When the endianness is in the default configuration, this address is for the least-significant 32 bits. The most-significant 32 bits can be accessed at this address +4h.

10.13.153.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RWDOG															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RWDOG															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.153.3 Fields

Field	Description
63-32 —	Reserved
31-0 RWDOG	Run Time Watchdog Time-Out value. This holds the starting value of the RTIC Run Time Watchdog Timer.

10.13.154 RTIC Endian Register (REND)

The RTIC Endian Register is used to allow for data ordering corrections when data is not retrieved from external memory in the proper order. These data ordering corrections are most likely to be needed on a mixed endian platform. 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.

10.13.154.1 Offset

Register	Offset
REND	6034h

10.13.154.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				RHWS				RBS				REPO			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.154.3 Fields

Field	Description
31-12 —	Reserved
11-8 RHWS	RTIC Half-Word Swap. This allows for a software controllable half-word swap in the DMA to assist with mixed Endianess platforms. This may be necessary if message data is not swapped properly when accessing memories. The word 01234567h placed in memory will become 45670123h when written into the hashing engine. The memory blocks are configured as follows: 1xxx - Half-Word Swap Memory Block D x1xx - Half-Word Swap Memory Block C xx1x - Half-Word Swap Memory Block B xxx1 - Half-Word Swap Memory Block A
7-4 RBS	RTIC Byte Swap. This allows for a software controllable byte swap to assist with mixed Endianess platforms. This byte swap works in conjunction with the platform endian configuration indicated by the PLEND bit in the CAAM Status Register. The word 01234567h placed in memory becomes 67452301h when written into the hashing engine. The memory blocks are configured as follows:

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Byte Swap Bit</th><th>PLEND</th><th>WORD</th></tr>
<tr><td>0</td><td>0</td><td>67452301h</td></tr>
<tr><td>1</td><td>0</td><td>01234567h</td></tr>
<tr><td>0</td><td>1</td><td>01234567h</td></tr>
<tr><td>1</td><td>1</td><td>67452301h</td></tr>
</table>
1xxx - Byte Swap Memory Block D
x1xx - Byte Swap Memory Block C
xx1x - Byte Swap Memory Block B
xxx1 - Byte Swap Memory Block A
</td></tr>
<tr><td>3-0
REPO</td><td>RTIC Endian Platform Override. This allows for the current platform endian configuration bit (PLEND bit in the CAAM Status Register) to be overridden by bits in the REPO field. PLEND is either Big Endian =1 or Little Endian =0. Setting a REPO bit to 1 will cause the data read from the corresponding memory block to be interpreted as Big Endian if PLEND specifies Little Endian, or Little Endian if PLEND specifies Big Endian.
The memory blocks are configured as follows:
1xxx - Byte Swap Memory Block D
x1xx - Byte Swap Memory Block C
xx1x - Byte Swap Memory Block B
xxx1 - Byte Swap Memory Block A</td></tr>
</table>

10.13.155 RTIC Memory Block a Address b Register (RMAA0 - RMDA1)

For an explanation of the RTIC Memory Block registers, see RTIC Memory Block Address/Length Registers

The RTIC Memory Block a Address b Register (RMaAb) specifies the starting address of segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The length of data referred to by this pointer (see Address pointers.) is found in the RTIC Memory Block a Length b Register (RMaLb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writeable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section RTIC Control Register (RCTL)) for the corresponding memory block.

10.13.155.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

Register	Offset	Description
RMaAb	6100h + (a × 20h) + (b × 10h)	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.155.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MEMBLKADDR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MEMBLKADDR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.155.3 Fields

Field	Description
63-32 —	Reserved
31-0 MEMBLKADDR	Memory Block Address. The MEMBLKADDR field of RMaAb holds the starting address of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D).

10.13.156 RTIC Memory Block a Length b Register (RMAL0 - RMDL1)

For an explanation of the RTIC Memory Block registers, see RTIC Memory Block Address/Length Registers

The RTIC Memory Block a Length b Register (RMaLb) specifies the number of bytes to hash in segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The starting address of segment b of RTIC Memory Block a is specified in the RTIC Memory Block a Address b Register (RMaAb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writeable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section RTIC Control Register (RCTL)) for the corresponding memory block.

Note that programming a memory segment (A, B, C, D) to have a zero length (length_1 and length_2) will cause RTIC to generate a bad descriptor.

In RTIC versions RMJV= 0 and RMNV <=1 this can be detected by means of a watchdog timer. In hash-once operation this will be detected only if the DECO watchdog timer is enabled. This will cause the descriptor that is programmed by RTIC to be detected by the watchdog and flagged as an Address Error in the status register. In run-time operation the bad descriptor will be flagged by either the RTIC watchdog timer or the DECO watchdog timer. If the RTIC watchdog timer detects this condition then it will be flagged as an RTIC Watchdog Error. If instead the DECO watchdog catches it, then it will be flagged as an Address Error.

In later versions of RTIC bad RTIC descriptors will be flagged immediately as Address Errors.

10.13.156.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

Register	Offset
RMaLb	610Ch + (a × 20h) + (b × 10h)

10.13.156.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MEMBLKLEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MEMBLKLEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.156.3 Fields

Field	Description
31-0 MEMBLKLEN	Memory Block Lengths. The MEMBLKLEN field of RMaLb holds the length, in bytes, of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D).

10.13.157 RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

The results of the RTIC hashing operations are stored in the RTIC Hash Result Registers (256 bits for SHA-256, 512 bits for SHA-512). The hash result for Memory Block a (a= A,B,C,D) is accessed in contiguous word addresses beginning at the base address of RTIC Hash Result Register a. For each Memory Block, there are 2 addresses associated with RTIC Hash Result Register a. Reading successive words starting at the RaMDB address will return successive words, in big endian format, of the hash result for Memory Block a. Reading successive words starting at the RaMDL address will return successive words, in little endian format, of the hash result for Memory Block a.

10.13.157.1 Offset

Register	Offset	Description
RAMDB_0	6200h	RTIC Mem Block A Hash Result Big Endian Format Word 0
RAMDB_1	6204h	RTIC Mem Block A Hash Result Big Endian Format Word 1
RAMDB_2	6208h	RTIC Mem Block A Hash Result Big Endian Format Word 2
RAMDB_3	620Ch	RTIC Mem Block A Hash Result Big Endian Format Word 3
RAMDB_4	6210h	RTIC Mem Block A Hash Result Big Endian Format Word 4
RAMDB_5	6214h	RTIC Mem Block A Hash Result Big Endian Format Word 5
RAMDB_6	6218h	RTIC Mem Block A Hash Result Big Endian Format Word 6
RAMDB_7	621Ch	RTIC Mem Block A Hash Result Big Endian Format Word 7
RAMDB_8	6220h	RTIC Mem Block A Hash Result Big Endian Format Word 8
RAMDB_9	6224h	RTIC Mem Block A Hash Result Big Endian Format Word 9
RAMDB_10	6228h	RTIC Mem Block A Hash Result Big Endian Format Word 10
RAMDB_11	622Ch	RTIC Mem Block A Hash Result Big Endian Format Word 11
RAMDB_12	6230h	RTIC Mem Block A Hash Result Big Endian Format Word 12
RAMDB_13	6234h	RTIC Mem Block A Hash Result Big Endian Format Word 13
RAMDB_14	6238h	RTIC Mem Block A Hash Result Big Endian Format Word 14
RAMDB_15	623Ch	RTIC Mem Block A Hash Result Big Endian Format Word 15
RAMDB_16	6240h	RTIC Mem Block A Hash Result Big Endian Format Word 16
RAMDB_17	6244h	RTIC Mem Block A Hash Result Big Endian Format Word 17
RAMDB_18	6248h	RTIC Mem Block A Hash Result Big Endian Format Word 18
RAMDB_19	624Ch	RTIC Mem Block A Hash Result Big Endian Format Word 19
RAMDB_20	6250h	RTIC Mem Block A Hash Result Big Endian Format Word 20
RAMDB_21	6254h	RTIC Mem Block A Hash Result Big Endian Format Word 21

Table continues on the next page...

Register	Offset	Description
RAMDB_22	6258h	RTIC Mem Block A Hash Result Big Endian Format Word 22
RAMDB_23	625Ch	RTIC Mem Block A Hash Result Big Endian Format Word 23
RAMDB_24	6260h	RTIC Mem Block A Hash Result Big Endian Format Word 24
RAMDB_25	6264h	RTIC Mem Block A Hash Result Big Endian Format Word 25
RAMDB_26	6268h	RTIC Mem Block A Hash Result Big Endian Format Word 26
RAMDB_27	626Ch	RTIC Mem Block A Hash Result Big Endian Format Word 27
RAMDB_28	6270h	RTIC Mem Block A Hash Result Big Endian Format Word 28
RAMDB_29	6274h	RTIC Mem Block A Hash Result Big Endian Format Word 29
RAMDB_30	6278h	RTIC Mem Block A Hash Result Big Endian Format Word 30
RAMDB_31	627Ch	RTIC Mem Block A Hash Result Big Endian Format Word 31
RAMDL_0	6280h	RTIC Mem Block A Hash Result Little Endian Format Word 0
RAMDL_1	6284h	RTIC Mem Block A Hash Result Little Endian Format Word 1
RAMDL_2	6288h	RTIC Mem Block A Hash Result Little Endian Format Word 2
RAMDL_3	628Ch	RTIC Mem Block A Hash Result Little Endian Format Word 3
RAMDL_4	6290h	RTIC Mem Block A Hash Result Little Endian Format Word 4
RAMDL_5	6294h	RTIC Mem Block A Hash Result Little Endian Format Word 5
RAMDL_6	6298h	RTIC Mem Block A Hash Result Little Endian Format Word 6
RAMDL_7	629Ch	RTIC Mem Block A Hash Result Little Endian Format Word 7
RAMDL_8	62A0h	RTIC Mem Block A Hash Result Little Endian Format Word 8
RAMDL_9	62A4h	RTIC Mem Block A Hash Result Little Endian Format Word 9
RAMDL_10	62A8h	RTIC Mem Block A Hash Result Little Endian Format Word 10
RAMDL_11	62ACh	RTIC Mem Block A Hash Result Little Endian Format Word 11
RAMDL_12	62B0h	RTIC Mem Block A Hash Result Little Endian Format Word 12
RAMDL_13	62B4h	RTIC Mem Block A Hash Result Little Endian Format Word 13

Table continues on the next page...

Register	Offset	Description
RAMDL_14	62B8h	RTIC Mem Block A Hash Result Little Endian Format Word 14
RAMDL_15	62BCh	RTIC Mem Block A Hash Result Little Endian Format Word 15
RAMDL_16	62C0h	RTIC Mem Block A Hash Result Little Endian Format Word 16
RAMDL_17	62C4h	RTIC Mem Block A Hash Result Little Endian Format Word 17
RAMDL_18	62C8h	RTIC Mem Block A Hash Result Little Endian Format Word 18
RAMDL_19	62CCh	RTIC Mem Block A Hash Result Little Endian Format Word 19
RAMDL_20	62D0h	RTIC Mem Block A Hash Result Little Endian Format Word 20
RAMDL_21	62D4h	RTIC Mem Block A Hash Result Little Endian Format Word 21
RAMDL_22	62D8h	RTIC Mem Block A Hash Result Little Endian Format Word 22
RAMDL_23	62DCh	RTIC Mem Block A Hash Result Little Endian Format Word 23
RAMDL_24	62E0h	RTIC Mem Block A Hash Result Little Endian Format Word 24
RAMDL_25	62E4h	RTIC Mem Block A Hash Result Little Endian Format Word 25
RAMDL_26	62E8h	RTIC Mem Block A Hash Result Little Endian Format Word 26
RAMDL_27	62ECh	RTIC Mem Block A Hash Result Little Endian Format Word 27
RAMDL_28	62F0h	RTIC Mem Block A Hash Result Little Endian Format Word 28
RAMDL_29	62F4h	RTIC Mem Block A Hash Result Little Endian Format Word 29
RAMDL_30	62F8h	RTIC Mem Block A Hash Result Little Endian Format Word 30
RAMDL_31	62FCh	RTIC Mem Block A Hash Result Little Endian Format Word 31
RBMDB_0	6300h	RTIC Mem Block B Hash Result Big Endian Format Word 0
RBMDB_1	6304h	RTIC Mem Block B Hash Result Big Endian Format Word 1
RBMDB_2	6308h	RTIC Mem Block B Hash Result Big Endian Format Word 2
RBMDB_3	630Ch	RTIC Mem Block B Hash Result Big Endian Format Word 3
RBMDB_4	6310h	RTIC Mem Block B Hash Result Big Endian Format Word 4
RBMDB_5	6314h	RTIC Mem Block B Hash Result Big Endian Format Word 5

Table continues on the next page...

Register	Offset	Description
RBMDB_6	6318h	RTIC Mem Block B Hash Result Big Endian Format Word 6
RBMDB_7	631Ch	RTIC Mem Block B Hash Result Big Endian Format Word 7
RBMDB_8	6320h	RTIC Mem Block B Hash Result Big Endian Format Word 8
RBMDB_9	6324h	RTIC Mem Block B Hash Result Big Endian Format Word 9
RBMDB_10	6328h	RTIC Mem Block B Hash Result Big Endian Format Word 10
RBMDB_11	632Ch	RTIC Mem Block B Hash Result Big Endian Format Word 11
RBMDB_12	6330h	RTIC Mem Block B Hash Result Big Endian Format Word 12
RBMDB_13	6334h	RTIC Mem Block B Hash Result Big Endian Format Word 13
RBMDB_14	6338h	RTIC Mem Block B Hash Result Big Endian Format Word 14
RBMDB_15	633Ch	RTIC Mem Block B Hash Result Big Endian Format Word 15
RBMDB_16	6340h	RTIC Mem Block B Hash Result Big Endian Format Word 16
RBMDB_17	6344h	RTIC Mem Block B Hash Result Big Endian Format Word 17
RBMDB_18	6348h	RTIC Mem Block B Hash Result Big Endian Format Word 18
RBMDB_19	634Ch	RTIC Mem Block B Hash Result Big Endian Format Word 19
RBMDB_20	6350h	RTIC Mem Block B Hash Result Big Endian Format Word 20
RBMDB_21	6354h	RTIC Mem Block B Hash Result Big Endian Format Word 21
RBMDB_22	6358h	RTIC Mem Block B Hash Result Big Endian Format Word 22
RBMDB_23	635Ch	RTIC Mem Block B Hash Result Big Endian Format Word 23
RBMDB_24	6360h	RTIC Mem Block B Hash Result Big Endian Format Word 24
RBMDB_25	6364h	RTIC Mem Block B Hash Result Big Endian Format Word 25
RBMDB_26	6368h	RTIC Mem Block B Hash Result Big Endian Format Word 26
RBMDB_27	636Ch	RTIC Mem Block B Hash Result Big Endian Format Word 27
RBMDB_28	6370h	RTIC Mem Block B Hash Result Big Endian Format Word 28
RBMDB_29	6374h	RTIC Mem Block B Hash Result Big Endian Format Word 29

Table continues on the next page...

Register	Offset	Description
RBMDB_30	6378h	RTIC Mem Block B Hash Result Big Endian Format Word 30
RBMDB_31	637Ch	RTIC Mem Block B Hash Result Big Endian Format Word 31
RBMDL_0	6380h	RTIC Mem Block B Hash Result Little Endian Format Word 0
RBMDL_1	6384h	RTIC Mem Block B Hash Result Little Endian Format Word 1
RBMDL_2	6388h	RTIC Mem Block B Hash Result Little Endian Format Word 2
RBMDL_3	638Ch	RTIC Mem Block B Hash Result Little Endian Format Word 3
RBMDL_4	6390h	RTIC Mem Block B Hash Result Little Endian Format Word 4
RBMDL_5	6394h	RTIC Mem Block B Hash Result Little Endian Format Word 5
RBMDL_6	6398h	RTIC Mem Block B Hash Result Little Endian Format Word 6
RBMDL_7	639Ch	RTIC Mem Block B Hash Result Little Endian Format Word 7
RBMDL_8	63A0h	RTIC Mem Block B Hash Result Little Endian Format Word 8
RBMDL_9	63A4h	RTIC Mem Block B Hash Result Little Endian Format Word 9
RBMDL_10	63A8h	RTIC Mem Block B Hash Result Little Endian Format Word 10
RBMDL_11	63ACh	RTIC Mem Block B Hash Result Little Endian Format Word 11
RBMDL_12	63B0h	RTIC Mem Block B Hash Result Little Endian Format Word 12
RBMDL_13	63B4h	RTIC Mem Block B Hash Result Little Endian Format Word 13
RBMDL_14	63B8h	RTIC Mem Block B Hash Result Little Endian Format Word 14
RBMDL_15	63BCh	RTIC Mem Block B Hash Result Little Endian Format Word 15
RBMDL_16	63C0h	RTIC Mem Block B Hash Result Little Endian Format Word 16
RBMDL_17	63C4h	RTIC Mem Block B Hash Result Little Endian Format Word 17
RBMDL_18	63C8h	RTIC Mem Block B Hash Result Little Endian Format Word 18
RBMDL_19	63CCh	RTIC Mem Block B Hash Result Little Endian Format Word 19
RBMDL_20	63D0h	RTIC Mem Block B Hash Result Little Endian Format Word 20
RBMDL_21	63D4h	RTIC Mem Block B Hash Result Little Endian Format Word 21

Table continues on the next page...

Register	Offset	Description
RBMDL_22	63D8h	RTIC Mem Block B Hash Result Little Endian Format Word 22
RBMDL_23	63DCh	RTIC Mem Block B Hash Result Little Endian Format Word 23
RBMDL_24	63E0h	RTIC Mem Block B Hash Result Little Endian Format Word 24
RBMDL_25	63E4h	RTIC Mem Block B Hash Result Little Endian Format Word 25
RBMDL_26	63E8h	RTIC Mem Block B Hash Result Little Endian Format Word 26
RBMDL_27	63ECh	RTIC Mem Block B Hash Result Little Endian Format Word 27
RBMDL_28	63F0h	RTIC Mem Block B Hash Result Little Endian Format Word 28
RBMDL_29	63F4h	RTIC Mem Block B Hash Result Little Endian Format Word 29
RBMDL_30	63F8h	RTIC Mem Block B Hash Result Little Endian Format Word 30
RBMDL_31	63FCh	RTIC Mem Block B Hash Result Little Endian Format Word 31
RCMDB_0	6400h	RTIC Mem Block C Hash Result Big Endian Format Word 0
RCMDB_1	6404h	RTIC Mem Block C Hash Result Big Endian Format Word 1
RCMDB_2	6408h	RTIC Mem Block C Hash Result Big Endian Format Word 2
RCMDB_3	640Ch	RTIC Mem Block C Hash Result Big Endian Format Word 3
RCMDB_4	6410h	RTIC Mem Block C Hash Result Big Endian Format Word 4
RCMDB_5	6414h	RTIC Mem Block C Hash Result Big Endian Format Word 5
RCMDB_6	6418h	RTIC Mem Block C Hash Result Big Endian Format Word 6
RCMDB_7	641Ch	RTIC Mem Block C Hash Result Big Endian Format Word 7
RCMDB_8	6420h	RTIC Mem Block C Hash Result Big Endian Format Word 8
RCMDB_9	6424h	RTIC Mem Block C Hash Result Big Endian Format Word 9
RCMDB_10	6428h	RTIC Mem Block C Hash Result Big Endian Format Word 10
RCMDB_11	642Ch	RTIC Mem Block C Hash Result Big Endian Format Word 11
RCMDB_12	6430h	RTIC Mem Block C Hash Result Big Endian Format Word 12
RCMDB_13	6434h	RTIC Mem Block C Hash Result Big Endian Format Word 13

Table continues on the next page...

Register	Offset	Description
RCMDB_14	6438h	RTIC Mem Block C Hash Result Big Endian Format Word 14
RCMDB_15	643Ch	RTIC Mem Block C Hash Result Big Endian Format Word 15
RCMDB_16	6440h	RTIC Mem Block C Hash Result Big Endian Format Word 16
RCMDB_17	6444h	RTIC Mem Block C Hash Result Big Endian Format Word 17
RCMDB_18	6448h	RTIC Mem Block C Hash Result Big Endian Format Word 18
RCMDB_19	644Ch	RTIC Mem Block C Hash Result Big Endian Format Word 19
RCMDB_20	6450h	RTIC Mem Block C Hash Result Big Endian Format Word 20
RCMDB_21	6454h	RTIC Mem Block C Hash Result Big Endian Format Word 21
RCMDB_22	6458h	RTIC Mem Block C Hash Result Big Endian Format Word 22
RCMDB_23	645Ch	RTIC Mem Block C Hash Result Big Endian Format Word 23
RCMDB_24	6460h	RTIC Mem Block C Hash Result Big Endian Format Word 24
RCMDB_25	6464h	RTIC Mem Block C Hash Result Big Endian Format Word 25
RCMDB_26	6468h	RTIC Mem Block C Hash Result Big Endian Format Word 26
RCMDB_27	646Ch	RTIC Mem Block C Hash Result Big Endian Format Word 27
RCMDB_28	6470h	RTIC Mem Block C Hash Result Big Endian Format Word 28
RCMDB_29	6474h	RTIC Mem Block C Hash Result Big Endian Format Word 29
RCMDB_30	6478h	RTIC Mem Block C Hash Result Big Endian Format Word 30
RCMDB_31	647Ch	RTIC Mem Block C Hash Result Big Endian Format Word 31
RCMDL_0	6480h	RTIC Mem Block C Hash Result Little Endian Format Word 0
RCMDL_1	6484h	RTIC Mem Block C Hash Result Little Endian Format Word 1
RCMDL_2	6488h	RTIC Mem Block C Hash Result Little Endian Format Word 2
RCMDL_3	648Ch	RTIC Mem Block C Hash Result Little Endian Format Word 3
RCMDL_4	6490h	RTIC Mem Block C Hash Result Little Endian Format Word 4
RCMDL_5	6494h	RTIC Mem Block C Hash Result Little Endian Format Word 5

Table continues on the next page...

Register	Offset	Description
RCMDL_6	6498h	RTIC Mem Block C Hash Result Little Endian Format Word 6
RCMDL_7	649Ch	RTIC Mem Block C Hash Result Little Endian Format Word 7
RCMDL_8	64A0h	RTIC Mem Block C Hash Result Little Endian Format Word 8
RCMDL_9	64A4h	RTIC Mem Block C Hash Result Little Endian Format Word 9
RCMDL_10	64A8h	RTIC Mem Block C Hash Result Little Endian Format Word 10
RCMDL_11	64ACh	RTIC Mem Block C Hash Result Little Endian Format Word 11
RCMDL_12	64B0h	RTIC Mem Block C Hash Result Little Endian Format Word 12
RCMDL_13	64B4h	RTIC Mem Block C Hash Result Little Endian Format Word 13
RCMDL_14	64B8h	RTIC Mem Block C Hash Result Little Endian Format Word 14
RCMDL_15	64BCh	RTIC Mem Block C Hash Result Little Endian Format Word 15
RCMDL_16	64C0h	RTIC Mem Block C Hash Result Little Endian Format Word 16
RCMDL_17	64C4h	RTIC Mem Block C Hash Result Little Endian Format Word 17
RCMDL_18	64C8h	RTIC Mem Block C Hash Result Little Endian Format Word 18
RCMDL_19	64CCh	RTIC Mem Block C Hash Result Little Endian Format Word 19
RCMDL_20	64D0h	RTIC Mem Block C Hash Result Little Endian Format Word 20
RCMDL_21	64D4h	RTIC Mem Block C Hash Result Little Endian Format Word 21
RCMDL_22	64D8h	RTIC Mem Block C Hash Result Little Endian Format Word 22
RCMDL_23	64DCh	RTIC Mem Block C Hash Result Little Endian Format Word 23
RCMDL_24	64E0h	RTIC Mem Block C Hash Result Little Endian Format Word 24
RCMDL_25	64E4h	RTIC Mem Block C Hash Result Little Endian Format Word 25
RCMDL_26	64E8h	RTIC Mem Block C Hash Result Little Endian Format Word 26
RCMDL_27	64ECh	RTIC Mem Block C Hash Result Little Endian Format Word 27
RCMDL_28	64F0h	RTIC Mem Block C Hash Result Little Endian Format Word 28
RCMDL_29	64F4h	RTIC Mem Block C Hash Result Little Endian Format Word 29

Table continues on the next page...

Register	Offset	Description
RCMDL_30	64F8h	RTIC Mem Block C Hash Result Little Endian Format Word 30
RCMDL_31	64FCh	RTIC Mem Block C Hash Result Little Endian Format Word 31
RDMDB_0	6500h	RTIC Mem Block D Hash Result Big Endian Format Word 0
RDMDB_1	6504h	RTIC Mem Block D Hash Result Big Endian Format Word 1
RDMDB_2	6508h	RTIC Mem Block D Hash Result Big Endian Format Word 2
RDMDB_3	650Ch	RTIC Mem Block D Hash Result Big Endian Format Word 3
RDMDB_4	6510h	RTIC Mem Block D Hash Result Big Endian Format Word 4
RDMDB_5	6514h	RTIC Mem Block D Hash Result Big Endian Format Word 5
RDMDB_6	6518h	RTIC Mem Block D Hash Result Big Endian Format Word 6
RDMDB_7	651Ch	RTIC Mem Block D Hash Result Big Endian Format Word 7
RDMDB_8	6520h	RTIC Mem Block D Hash Result Big Endian Format Word 8
RDMDB_9	6524h	RTIC Mem Block D Hash Result Big Endian Format Word 9
RDMDB_10	6528h	RTIC Mem Block D Hash Result Big Endian Format Word 10
RDMDB_11	652Ch	RTIC Mem Block D Hash Result Big Endian Format Word 11
RDMDB_12	6530h	RTIC Mem Block D Hash Result Big Endian Format Word 12
RDMDB_13	6534h	RTIC Mem Block D Hash Result Big Endian Format Word 13
RDMDB_14	6538h	RTIC Mem Block D Hash Result Big Endian Format Word 14
RDMDB_15	653Ch	RTIC Mem Block D Hash Result Big Endian Format Word 15
RDMDB_16	6540h	RTIC Mem Block D Hash Result Big Endian Format Word 16
RDMDB_17	6544h	RTIC Mem Block D Hash Result Big Endian Format Word 17
RDMDB_18	6548h	RTIC Mem Block D Hash Result Big Endian Format Word 18
RDMDB_19	654Ch	RTIC Mem Block D Hash Result Big Endian Format Word 19
RDMDB_20	6550h	RTIC Mem Block D Hash Result Big Endian Format Word 20
RDMDB_21	6554h	RTIC Mem Block D Hash Result Big Endian Format Word 21

Table continues on the next page...

Register	Offset	Description
RDMDB_22	6558h	RTIC Mem Block D Hash Result Big Endian Format Word 22
RDMDB_23	655Ch	RTIC Mem Block D Hash Result Big Endian Format Word 23
RDMDB_24	6560h	RTIC Mem Block D Hash Result Big Endian Format Word 24
RDMDB_25	6564h	RTIC Mem Block D Hash Result Big Endian Format Word 25
RDMDB_26	6568h	RTIC Mem Block D Hash Result Big Endian Format Word 26
RDMDB_27	656Ch	RTIC Mem Block D Hash Result Big Endian Format Word 27
RDMDB_28	6570h	RTIC Mem Block D Hash Result Big Endian Format Word 28
RDMDB_29	6574h	RTIC Mem Block D Hash Result Big Endian Format Word 29
RDMDB_30	6578h	RTIC Mem Block D Hash Result Big Endian Format Word 30
RDMDB_31	657Ch	RTIC Mem Block D Hash Result Big Endian Format Word 31
RDMDL_0	6580h	RTIC Mem Block D Hash Result Little Endian Format Word 0
RDMDL_1	6584h	RTIC Mem Block D Hash Result Little Endian Format Word 1
RDMDL_2	6588h	RTIC Mem Block D Hash Result Little Endian Format Word 2
RDMDL_3	658Ch	RTIC Mem Block D Hash Result Little Endian Format Word 3
RDMDL_4	6590h	RTIC Mem Block D Hash Result Little Endian Format Word 4
RDMDL_5	6594h	RTIC Mem Block D Hash Result Little Endian Format Word 5
RDMDL_6	6598h	RTIC Mem Block D Hash Result Little Endian Format Word 6
RDMDL_7	659Ch	RTIC Mem Block D Hash Result Little Endian Format Word 7
RDMDL_8	65A0h	RTIC Mem Block D Hash Result Little Endian Format Word 8
RDMDL_9	65A4h	RTIC Mem Block D Hash Result Little Endian Format Word 9
RDMDL_10	65A8h	RTIC Mem Block D Hash Result Little Endian Format Word 10
RDMDL_11	65ACh	RTIC Mem Block D Hash Result Little Endian Format Word 11
RDMDL_12	65B0h	RTIC Mem Block D Hash Result Little Endian Format Word 12
RDMDL_13	65B4h	RTIC Mem Block D Hash Result Little Endian Format Word 13

Table continues on the next page...

Register	Offset	Description
RDMDL_14	65B8h	RTIC Mem Block D Hash Result Little Endian Format Word 14
RDMDL_15	65BCh	RTIC Mem Block D Hash Result Little Endian Format Word 15
RDMDL_16	65C0h	RTIC Mem Block D Hash Result Little Endian Format Word 16
RDMDL_17	65C4h	RTIC Mem Block D Hash Result Little Endian Format Word 17
RDMDL_18	65C8h	RTIC Mem Block D Hash Result Little Endian Format Word 18
RDMDL_19	65CCh	RTIC Mem Block D Hash Result Little Endian Format Word 19
RDMDL_20	65D0h	RTIC Mem Block D Hash Result Little Endian Format Word 20
RDMDL_21	65D4h	RTIC Mem Block D Hash Result Little Endian Format Word 21
RDMDL_22	65D8h	RTIC Mem Block D Hash Result Little Endian Format Word 22
RDMDL_23	65DCh	RTIC Mem Block D Hash Result Little Endian Format Word 23
RDMDL_24	65E0h	RTIC Mem Block D Hash Result Little Endian Format Word 24
RDMDL_25	65E4h	RTIC Mem Block D Hash Result Little Endian Format Word 25
RDMDL_26	65E8h	RTIC Mem Block D Hash Result Little Endian Format Word 26
RDMDL_27	65ECh	RTIC Mem Block D Hash Result Little Endian Format Word 27
RDMDL_28	65F0h	RTIC Mem Block D Hash Result Little Endian Format Word 28
RDMDL_29	65F4h	RTIC Mem Block D Hash Result Little Endian Format Word 29
RDMDL_30	65F8h	RTIC Mem Block D Hash Result Little Endian Format Word 30
RDMDL_31	65FCh	RTIC Mem Block D Hash Result Little Endian Format Word 31

10.13.157.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RTIC_Hash_Result															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RTIC_Hash_Result															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.157.3 Fields

Field	Description
31-0 RTIC_Hash_Result	RTIC_Hash_Result

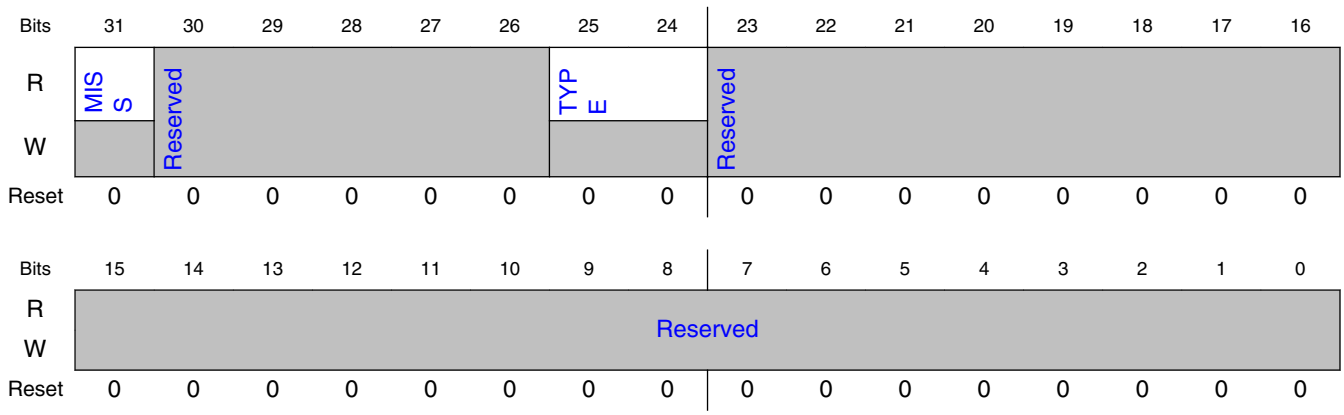
10.13.158 Recoverable Error Indication Record 0 for RTIC (REIR 0RTIC)

If a recoverable error occurs related to execution of a job from RTIC, error information will be captured in RTIC's REIR registers. Data for a second recoverable error related to jobs from RTIC will not be captured until the REIR0RTIC is written. If another bus error from RTIC occurs before then, the double error status bit (MISS) in REIR0RTIC will be set. When REIR0RTIC is written, all of RTIC's REIRRTIC registers are cleared and error capture is re-enabled.

10.13.158.1 Offset

Register	Offset
REIR0RTIC	6E00h

10.13.158.2 Diagram



10.13.158.3 Fields

Field	Description
31 MISS	If MISS=1, a second recoverable error associated with RTIC occurred before REIR0RTIC was written following a previous RTIC recoverable error.
30-26 —	Reserved
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 00b : reserved If TYPE = 01b : memory access error If TYPE = 10b : reserved If TYPE = 11b : reserved
23-0 —	Reserved

10.13.159 Recoverable Error Indication Record 2 for RTIC (REIR 2RTIC)

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

10.13.159.1 Offset

Register	Offset	Description
REIR2RTIC	6E08h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.159.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.159.3 Fields

Field	Description
63-0 ADDR	This register holds the address associated with the recoverable RTIC error.

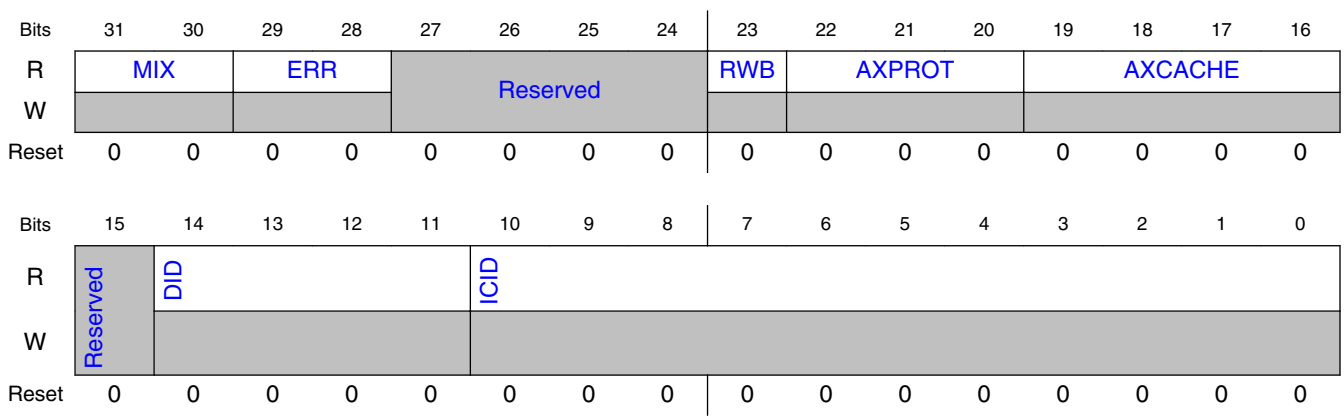
10.13.160 Recoverable Error Indication Record 4 for RTIC (REIR 4RTIC)

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

10.13.160.1 Offset

Register	Offset
REIR4RTIC	6E10h

10.13.160.2 Diagram



10.13.160.3 Fields

Field	Description
31-30 MIX	This field holds the memory interface index associated with the recoverable error.
29-28 ERR	This field holds the AXI error response associated with the recoverable error.
27-24 —	Reserved
23 RWB	This field specifies whether the memory access was a read or write.
22-20	This field holds the AXI protection transaction attribute used for the memory access.

Table continues on the next page...

Field	Description
AXPROT	
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15 —	Reserved
14-11 DID	This field holds the DID associated with the recoverable error.
10-0 ICID	This field holds the ICID associated with the recoverable error.

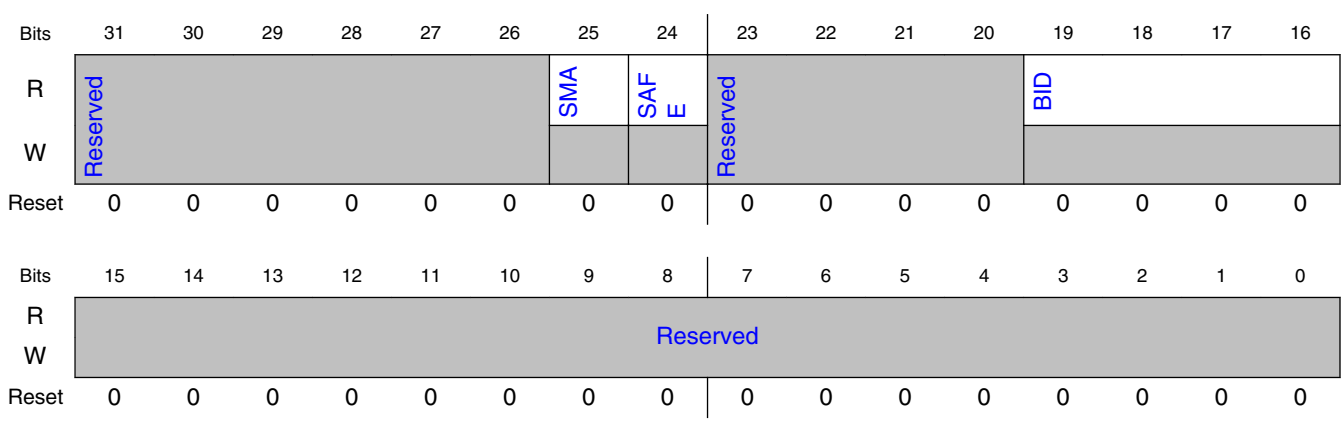
10.13.161 Recoverable Error Indication Record 5 for RTIC (REIR 5RTIC)

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

10.13.161.1 Offset

Register	Offset
REIR5RTIC	6E14h

10.13.161.2 Diagram



10.13.161.3 Fields

Field	Description
31-26 —	Reserved
25 SMA	This field indicates whether the bus transaction associated with the recoverable error was an attempted access to CAAM Secure Memory.
24 SAFE	SAFE indicates whether the AXI transaction associated with the recoverable error was a "safe" transaction.
23-20 —	Reserved
19-16 BID	This field holds the block identifier (see Table 10-243) of the source of the AXI transaction associated with the recoverable error.
15-0 —	Reserved

10.13.162 CCB 0 Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR)

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 has several independent definitions, one for Public Key algorithms (see Section CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)),one for RNG (see Section CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG)), and one for all others. 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.

This section defines the format of the Class 1 Mode Register when used with non-public-key algorithms. The Non-Public-Key algorithms are those that do not use the PKHA.

Some examples of how to build the Class 1 Mode Register for non-Public Key algorithms:

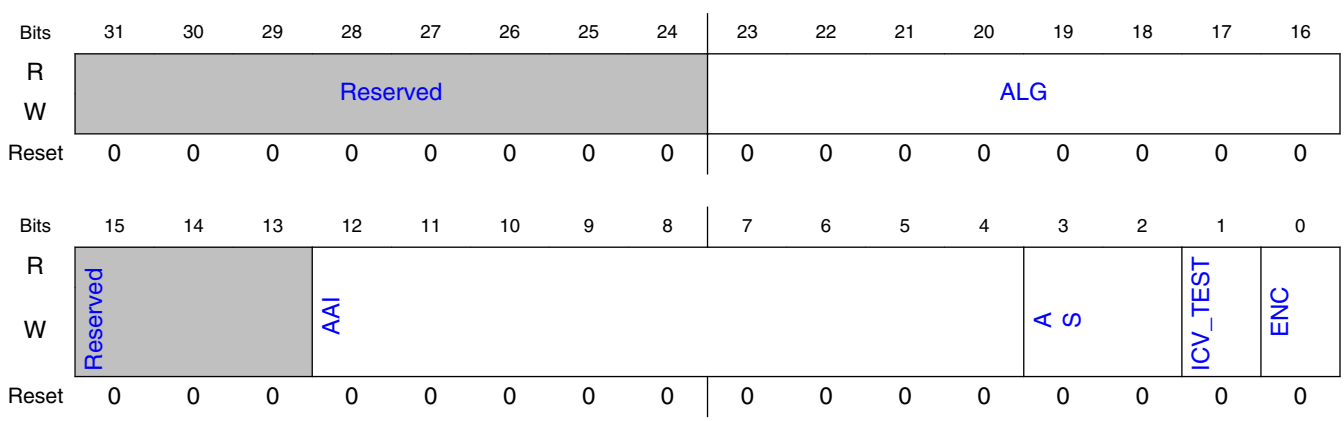
Table 10-246. Class 1 Mode Register examples for non-Public Key algorithms

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	8201090Eh	0001090Eh
AES Counter with mod=2^{128}	AES	CTR Modulus 2^{128}	--	no	Encrypt	82010001h	00010001h
Triple DES OFB mode with key parity	DES	OFB	--	no	Decrypt	82021400h	00021400h

10.13.162.1 Offset

Register	Offset	Description
C0C1MR	8004h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.162.2 Diagram



10.13.162.3 Fields

Field	Description
31-24	Reserved

Table continues on the next page...

Field	Description
—	
23-16 ALG	Algorithm. This field specifies which algorithm is being selected. 00010000 - AES 00100000 - DES 00100001 - 3DES 00110000 - ARC4 01010000 - RNG
15-13 —	Reserved
12-4 AAI	Additional Algorithm information. This field contains additional mode information associated with the executed algorithm. See also the section describing the appropriate CHA. For RNG OPERATION commands the AAI field is interpreted as shown in CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG). **NOTE:** Some algorithms do not require additional algorithm information and in those cases this field should be all 0s. The codes listed in the following table(s) are mutually exclusive, i.e., they cannot be ORed with each other.
3-2 AS	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 CCB 0 Class 1 Mode Register Format for RNG4 (C0C1 MR_RNG). 00 - Update

Table 10-247. AAI Interpretation for AES

Code	Interpretation		Code	Interpretation
00h	CTR (mod 2^{128})		80h	CCM (mod 2^{128}), [1]
10h	CBC		90h	GCM (mod 2^{32})[1]
20h	ECB			
30h	CFB128			
40h	OFB			
60h	CMAC			
70h	XCBC-MAC		12h	CBC_CS2 (CTS)
Note that for AES the MSB of AAI is the DK (Decrypt Key) bit. Setting the DK bit (i.e. ORing 100h with any AES code above) tells CAAM that the Key Register was loaded with the AES decrypt key, rather than the AES encrypt key. See the discussion in Differences between the AES encrypt and decrypt keys				

1. CCM and GCM use the key in the Class 1 Key Register.

Table 10-248. AAI Interpretation for DES

Code	Interpretation		Code	Interpretation
10h	CBC		30h	CFB8
20h	ECB		40h	OFB
80h ORed with any DES code above: Check odd parity				

1. CCM and GCM use the key in the Class 1 Key Register.

Table continues on the next page...

Field	Description
	01 - Initialize 10 - Finalize 11 - Initialize/Finalize
1 ICV_TEST	ICV Checking / Test AESA fault detection. (This is the definition of this bit for CHAs other than RNG. For the definition of this bit in RNG commands, see CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG)) For algorithms other than AESA 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. 0 - Don't compare 1 - Compare For AESA ECB mode: Test AESA fault detection In AESA ECB mode, this bit activates fault detection testing by injecting bit level errors into AESA core logic as defined in the first 128 bits of the context. 0 - Don't inject bit errors 1 - Inject bit errors
0 ENC	Encrypt/Decrypt. (This is the definition of this bit for CHAs other than RNG. For the definition of this bit in RNG commands, see CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG).) This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes. 0 - Decrypt. 1 - Encrypt.

1. CCM and GCM use the key in the Class 1 Key Register.

10.13.163 CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)

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 has several independent definitions, one for Public Key algorithms (see Section CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)),one for RNG (see Section CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG)), and one for all others. 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.

The following figure shows the Class 1 Mode Register format that is used with public key algorithms, which are algorithms that use PKHA. The Class 1 Mode register is automatically cleared following a PKHA Command. The bit assignments for the PKHA_MODE field shown in CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK) will be different depending on which of the three types of PKHA functions is being called. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION : clear memory function, Table PKHA OPERATION : Arithmetic Functions and Table PKHA OPERATION : copy memory functions.

10.13.163.1 Offset

Register	Offset	Description
C0C1MR_PK	8004h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.163.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved												PKHA_MODE_MS			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				PKHA_MODE_LS											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.163.3 Fields

Field	Description
31-20	Reserved

Table continues on the next page...

Field	Description
—	
19-16 PKHA_MODE_MS	PKHA_MODE most-significant 4 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION : clear memory function, Table PKHA OPERATION : Arithmetic Functions and Table PKHA OPERATION : copy memory functions.
15-12 —	Reserved
11-0 PKHA_MODE_LS	PKHA_MODE least significant 12 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION : clear memory function, Table PKHA OPERATION : Arithmetic Functions and Table PKHA OPERATION : copy memory functions.

10.13.164 CCB 0 Class 1 Mode Register Format for RNG4 (C0C1 MR_RNG)

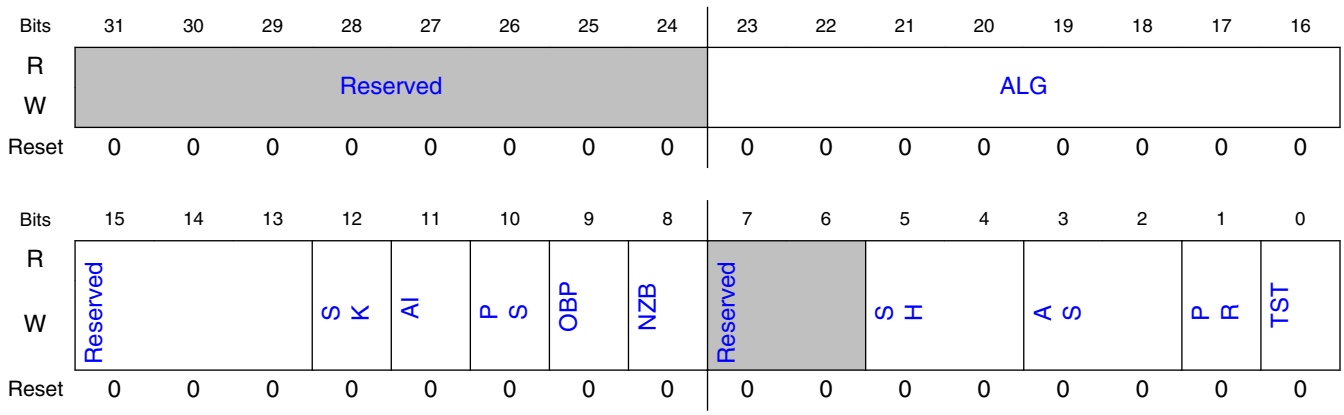
The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)),one for RNG (see Section CCB 0 Class 1 Mode Register Format for RNG4 (C0C1 MR_RNG)), and one for all others. 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.

When the Class 1 Mode register is used to control the RNG, the following format is used.

10.13.164.1 Offset

Register	Offset	Description
C0C1MR_RNG	8004h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.164.2 Diagram



10.13.164.3 Fields

Field	Description
31-24 —	Reserved
23-16 ALG	Algorithm. This field specifies which algorithm is being selected. 01010000 - RNG
15-13 —	Reserved
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. 0 - The destination for the RNG data is specified by the FIFO STORE command. 1 - The RNG data will go to the JDKEKR, TDKEKR and DSKR.
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). 0 - No additional entropy input has been provided. 1 - Additional entropy input has been provided.
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

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>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.
0 - No personalization string is included.
1 - A personalization string is included.</td></tr>
<tr><td>9
OBP</td><td>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.
0 - No odd byte parity.
1 - Generate random data with odd byte parity.</td></tr>
<tr><td>8
NZB</td><td>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=00h (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS!=00h.
0 - Generate random data with all-zero bytes permitted.
1 - Generate random data without any all-zero bytes.</td></tr>
<tr><td>7-6
—</td><td>Reserved. For RNG commands these bits of the AAI field are reserved.</td></tr>
<tr><td>5-4
SH</td><td>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</td></tr>
<tr><td>3-2
AS</td><td>Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown below:
<table>
<tr><th>AS Value</th><th>State Handle is already instantiated</th><th>State Handle is NOT already instantiated</th></tr>
<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 non-deterministic 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>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle.</td><td>Error</td></tr>
</table></td></tr>
</table>

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>1. 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 non-deterministic 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.</td></tr>
<tr><td>1
PR</td><td>Prediction Resistance. For RNG OPERATION commands this bit is interpreted as:

<table>
<tr><th>AS Value</th><th>PR = 0</th><th>PR = 1</th></tr>
<tr><td>00 Generate</td><td>Do NOT reseed prior to generating new random data</td><td>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.</td></tr>
<tr><td>01 Instantiate</td><td>Instantiate the state handle to NOT support prediction resistance</td><td>Instantiate the state handle to support prediction resistance</td></tr>
<tr><td>10 Reseed</td><td>Reseed the state handle. PR bit is ignored.</td><td>Reseed the state handle. PR bit is ignored.</td></tr>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle. PR bit is ignored.</td><td>Uninstantiate the state handle. PR bit is ignored.</td></tr>
</table>

1. 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 non-deterministic 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.</td></tr>
<tr><td>0
TST</td><td>Test Mode Request. For RNG OPERATION commands this bit is interpreted as:

<table>
<tr><th>AS Value</th><th>TST = 0</th><th>TST = 1</th></tr>
<tr><td>00 Generate</td><td>If the selected state handle is in non-deterministic mode, generate new random data.

If the selected state handle is in deterministic mode, generate a Test error.[-1]</td><td>If the selected state handle is in deterministic mode, generate new random data.

If the selected state handle is in non-deterministic mode, generate a Test error..</td></tr>
<tr><td>01 Instantiate</td><td>Instantiate the state handle in normal (non-deterministic) mode.</td><td>Instantiate the state handle in test (deterministic) mode.</td></tr>
<tr><td>10 Reseed</td><td>If the selected state handle is in non-deterministic mode, reseed the state handle.

If the selected state handle is in deterministic mode, generate a Test error.</td><td>If the selected state handle is in non-deterministic mode, reseed the state handle.

If the selected state handle is in deterministic mode, generate a Test error.</td></tr>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle. TST bit is ignored.</td><td>Uninstantiate the state handle. TST bit is ignored.</td></tr>
</table>
</td></tr>
</table>

Field	Description
	1. 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 non-deterministic 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.

1. 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 non-deterministic 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.

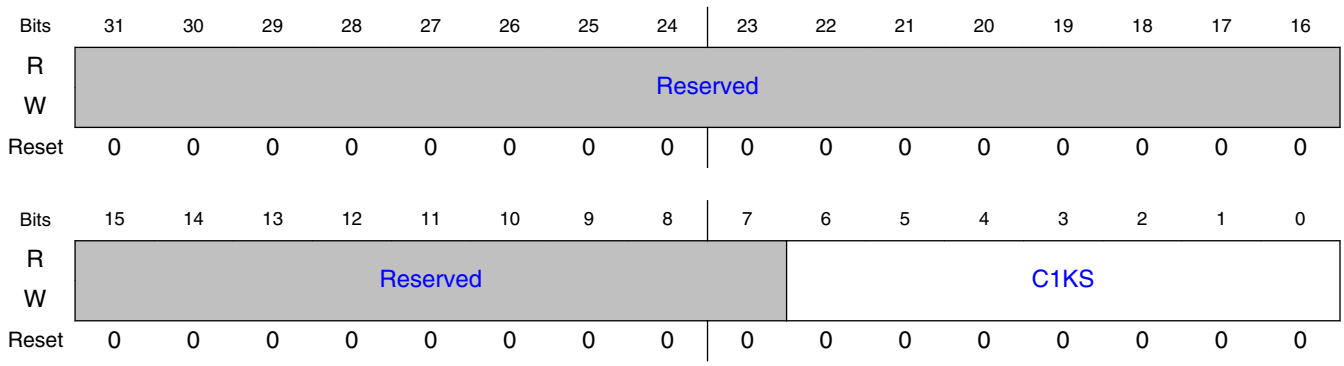
10.13.165 CCB 0 Class 1 Key Size Register (C0C1KSR)

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 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. When the PKHA E-RAM is loaded the PKHA E Size Register is automatically loaded with the correct size, rather than loading the Class 1 Key Size Register.

10.13.165.1 Offset

Register	Offset	Description
C0C1KSR	800Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.165.2 Diagram



10.13.165.3 Fields

Field	Description
31-7 —	Reserved
6-0 C1KS	Class 1 Key Size. This is the size of a Class 1 Key measured in bytes

10.13.166 CCB 0 Class 1 Data Size Register (C0C1DSR)

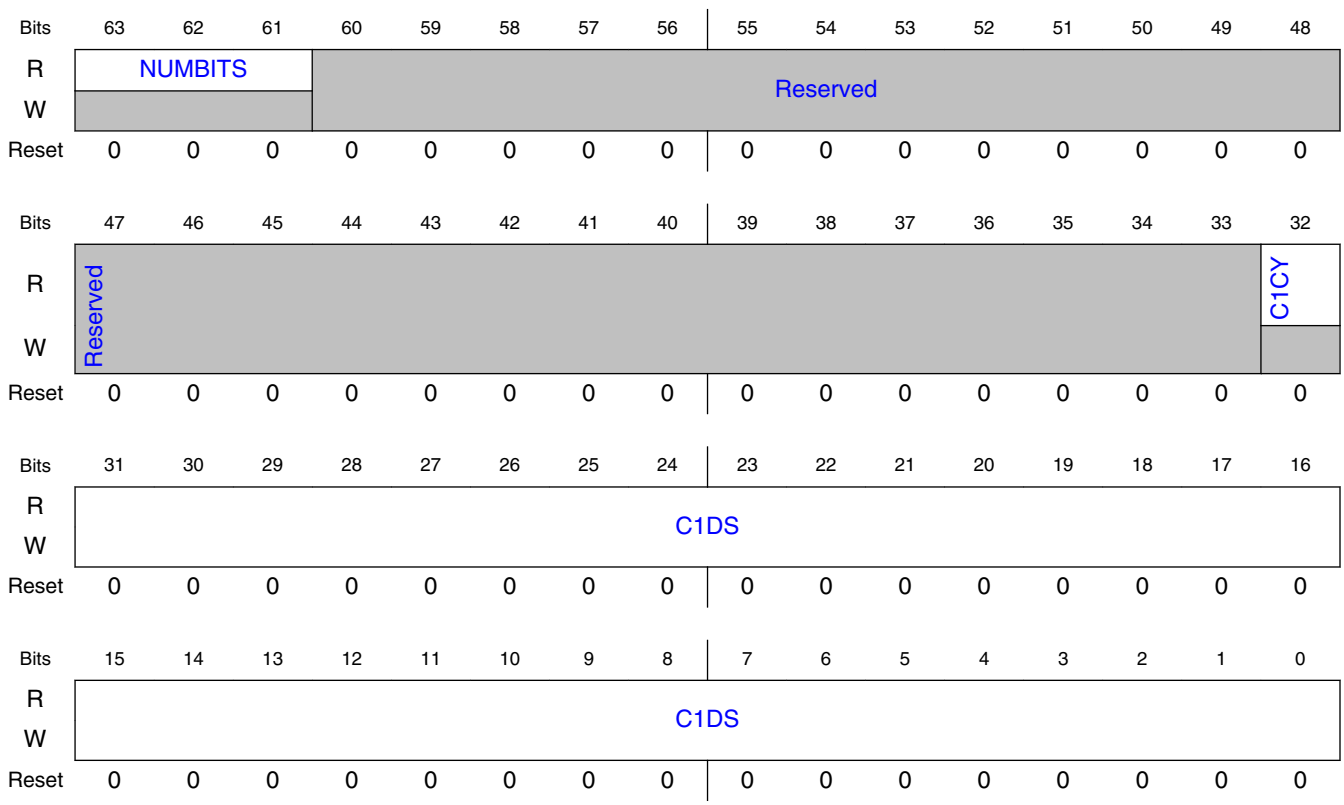
The Class 1 Data Size Register is used to tell the Class 1 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C1CY and C1DS fields to form a data size that is measured in bits. Note that writing to the C1DS 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.

10.13.166.1 Offset

Register	Offset	Description
C0C1DSR	8010h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.166.2 Diagram



10.13.166.3 Fields

Field	Description
63-61 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.
60-33 —	Reserved
32 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. 0 - No carry out of the C1 Data Size Reg. 1 - There was a carry out of the C1 Data Size Reg.
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.

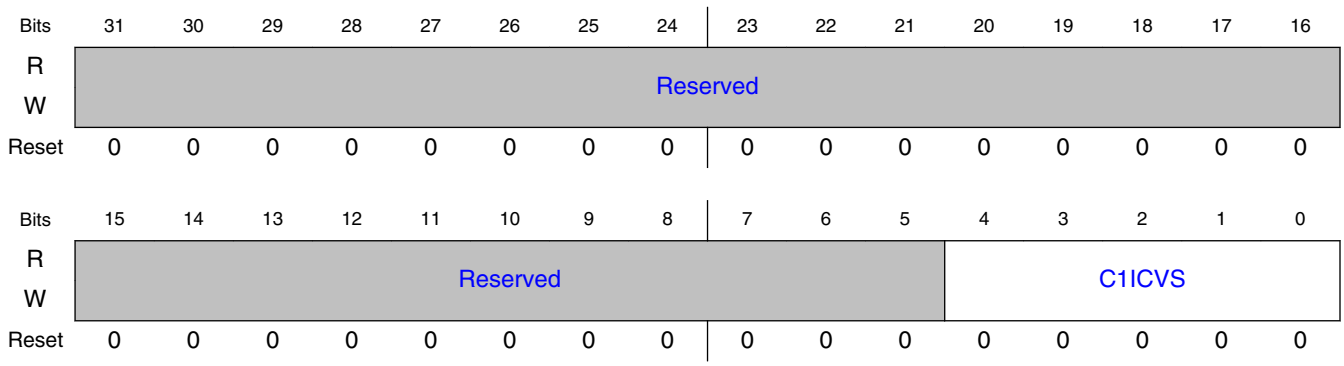
10.13.167 CCB 0 Class 1 ICV Size Register (C0C1ICVSR)

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-GCM). 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. FIFO LOAD commands can automatically load it when ICV is loaded.

10.13.167.1 Offset

Register	Offset	Description
C0C1ICVSR	801Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.167.2 Diagram



10.13.167.3 Fields

Field	Description
31-5 —	Reserved
4-0 C1ICVS	Class 1 ICV Size, in Bytes.

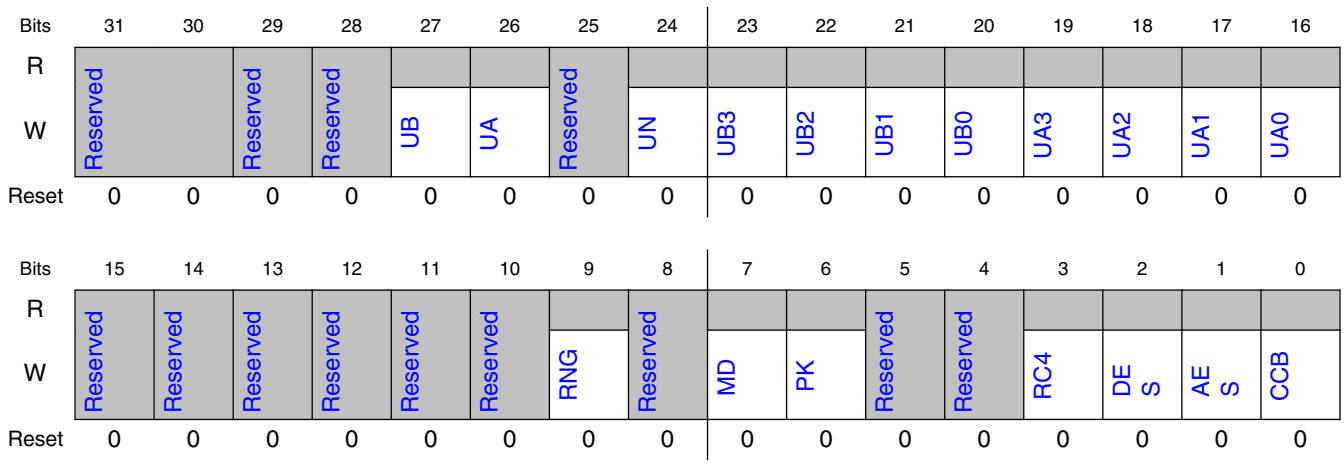
10.13.168 CCB 0 CHA Control Register (C0CCTRL)

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.

10.13.168.1 Offset

Register	Offset	Description
C0CCTRL	8034h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.168.2 Diagram



10.13.168.3 Fields

Field	Description
31-30 —	Reserved
29 —	Reserved
28 —	Reserved
27 UB	Unload the PKHA B Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B memory into the Output Data FIFO. 0 - Don't unload the PKHA B Memory. 1 - Unload the PKHA B Memory into OFIFO.
26 UA	Unload the PKHA A Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A memory into the Output Data FIFO. 0 - Don't unload the PKHA A Memory. 1 - Unload the PKHA A Memory into OFIFO.
25 —	Reserved
24 UN	Unload the PKHA N Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the N memory into the Output Data FIFO. 0 - Don't unload the PKHA N Memory. 1 - Unload the PKHA N Memory into OFIFO.
23 UB3	Unload the PKHA B3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B3 memory into the Output Data FIFO. 0 - Don't unload the PKHA B3 Memory.

Table continues on the next page...

Field	Description
	1 - Unload the PKHA B3 Memory into OFIFO.
22 UB2	Unload the PKHA B2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B2 memory into the Output Data FIFO. 0 - Don't unload the PKHA B2 Memory. 1 - Unload the PKHA B2 Memory into OFIFO.
21 UB1	Unload the PKHA B1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B1 memory into the Output Data FIFO. 0 - Don't unload the PKHA B1 Memory. 1 - Unload the PKHA B1 Memory into OFIFO.
20 UB0	Unload the PKHA B0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B0 memory into the Output Data FIFO. 0 - Don't unload the PKHA B0 Memory. 1 - Unload the PKHA B0 Memory into OFIFO.
19 UA3	Unload the PKHA A3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A3 memory into the Output Data FIFO. 0 - Don't unload the PKHA A3 Memory. 1 - Unload the PKHA A3 Memory into OFIFO.
18 UA2	Unload the PKHA A2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A2 memory into the Output Data FIFO. 0 - Don't unload the PKHA A2 Memory. 1 - Unload the PKHA A2 Memory into OFIFO.
17 UA1	Unload the PKHA A1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A1 memory into the Output Data FIFO. 0 - Don't unload the PKHA A1 Memory. 1 - Unload the PKHA A1 Memory into OFIFO.
16 UA0	Unload the PKHA A0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A0 memory into the Output Data FIFO. 0 - Don't unload the PKHA A0 Memory. 1 - Unload the PKHA A0 Memory into OFIFO.
15 —	Reserved
14 —	Reserved
13 —	Reserved
12 —	Reserved
11 —	Reserved
10 —	Reserved
9	Reset Random Number Generator. Writing a 1 to this bit resets the Random Number Generator.

Table continues on the next page...

Field	Description
RNG	0 - Do Not Reset 1 - Reset Random Number Generator Block.
8 —	Reserved
7 MD	Reset MDHA. Writing a 1 to this bit resets the Message Digest Hardware Accelerator. 0 - Do Not Reset 1 - Reset Message Digest Hardware Accelerator
6 PK	Reset PKHA. Writing a 1 to this bit resets the Public Key Hardware Accelerator. 0 - Do Not Reset 1 - Reset Public Key Hardware Accelerator
5 —	Reserved
4 —	Reserved
3 RC4	Reset AFHA. Writing a 1 to this bit resets the ARC4 Hardware Accelerator. 0 - Do Not Reset 1 - Reset ARC4 Hardware Accelerator
2 DES	Reset DESA. Writing a 1 to this bit resets the DES Accelerator. 0 - Do Not Reset 1 - Reset DES Accelerator
1 AES	Reset AESA. Writing a 1 to this bit resets the AES Accelerator. 0 - Do Not Reset 1 - Reset AES Accelerator
0 CCB	Reset CCB. Writing a 1 to this bit resets the CCB. If a CHA reset is required, the CHA should be reset after the CCB is reset to prevent automatic restart of the CHA due to a non-0 CCB Mode Register (see CnC1MR or CnC2MR) unless a CHA restart is intended. As an alternative to sequential reset of both CCB and CHA(s), consider setting both the CHA bit(s) and the CCB bit or using the CCB Clear Written Register. **NOTE:** In CAAM versions before era 6, this bit was called 'ALL' and it could be used to reset the CCB and all CHAs. While this was convenient, it also made the CHA available for sharing, which may not be what the application requires. 0 - Do Not Reset 1 - Reset CCB

10.13.169 CCB 0 Interrupt Control Register (C0ICTL)

The IRQ Control Register shows the status of all CCB "done" interrupts and "error" interrupts and provides controls for clearing these interrupts.

10.13.169.1 Offset

Register	Offset	Description
C0ICTL	803Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.169.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RNE_I	Reserved	MEI	PE_I	Reserved	Reserved	RCE_I	DE_I	AEI	Reserved
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RNDI	Reserved	MDI	PDI	Reserved	Reserved	RCDI	DDI	ADI	Reserved
W							W1C		W1C	W1C			W1C	W1C	W1C	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.169.3 Fields

Field	Description
31 —	Reserved
30 —	Reserved
29 —	Reserved
28 —	Reserved
27 —	Reserved
26 —	Reserved

Table continues on the next page...

Field	Description
25 RNEI	RNG Error Interrupt asserted. 0 - No RNG error detected 1 - RNG error detected
24 —	Reserved
23 MEI	MDHA (hashing) Error Interrupt asserted. 0 - No MDHA error detected 1 - MDHA error detected
22 PEI	PKHA (Public Key) Error Interrupt asserted. 0 - No PKHA error detected 1 - PKHA error detected
21 —	Reserved
20 —	Reserved
19 RCEI	AFHA (ARC4) Error Interrupt asserted. 0 - No AFHA error detected 1 - AFHA error detected
18 DEI	DESA Error Interrupt asserted. 0 - No DESA error detected 1 - DESA error detected
17 AEI	AESA Error Interrupt asserted. 0 - No AESA error detected 1 - AESA error detected
16 —	Reserved
15 —	Reserved
14 —	Reserved
13 —	Reserved
12 —	Reserved
11 —	Reserved
10 —	Reserved
9	RNG done interrupt.

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RNDI</td><td>
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>RNG Done Interrupt asserted</td><td>Clear the RNG Done Interrupt</td></tr>
</table>
</td></tr>
<tr><td>8
—</td><td>Reserved</td></tr>
<tr><td>7
MDI</td><td>MDHA (hashing) done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>MDHA Done Interrupt asserted</td><td>Clear the MDHA Done Interrupt</td></tr>
</table>
</td></tr>
<tr><td>6
PDI</td><td>PKHA (Public Key) done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>PKHA Done Interrupt asserted</td><td>Clear the PKHA Done Interrupt</td></tr>
</table>
</td></tr>
<tr><td>5
—</td><td>Reserved</td></tr>
<tr><td>4
—</td><td>Reserved</td></tr>
<tr><td>3
RCDI</td><td>ARC4 done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>ARC4 Done Interrupt asserted</td><td>Clear the ARC4 Done Interrupt</td></tr>
</table>
</td></tr>
<tr><td>2
DDI</td><td>DESA done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>DESA Done Interrupt asserted</td><td>Clear the DESA Done Interrupt</td></tr>
</table>
</td></tr>
<tr><td>1
ADI</td><td>AESA done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>AESA Done Interrupt asserted</td><td>Clear the AESA Done Interrupt</td></tr>
</table>
</td></tr>
</table>

Table continues on the next page...

Field	Description
0 —	Reserved

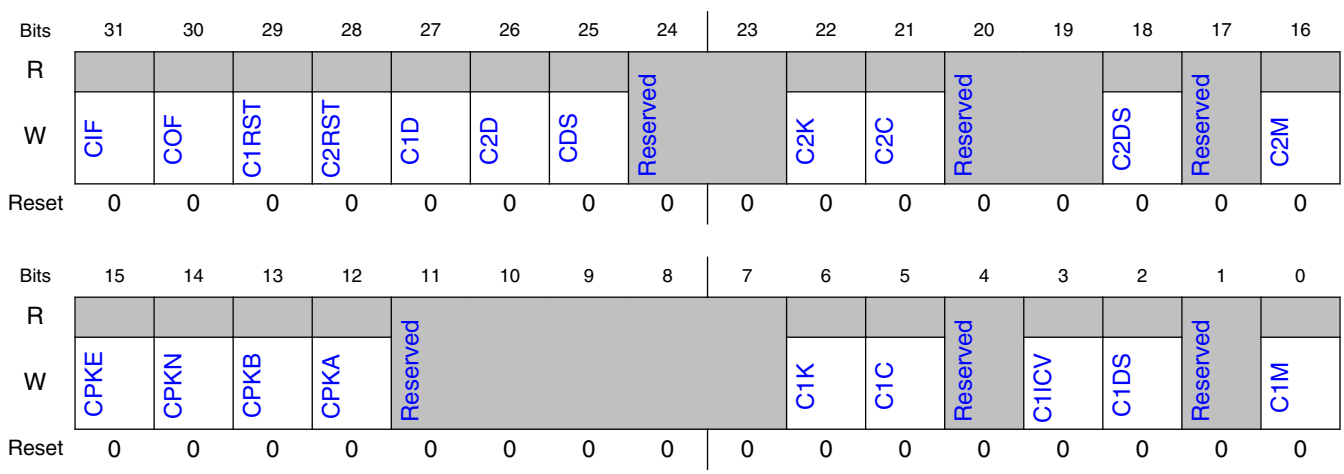
10.13.170 CCB 0 Clear Written Register (C0CWR)

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.

10.13.170.1 Offset

Register	Offset	Description
C0CWR	8044h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.170.2 Diagram



10.13.170.3 Fields

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. 0 - Don't clear the IFIFO. 1 - Clear the IFIFO.
30 COF	Clear Output FIFO. Writing a 1 to this bit causes the Output FIFO to be cleared. 0 - Don't clear the OFIFO. 1 - Clear the OFIFO.
29 C1RST	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. 0 - Don't reset the Class 1 CHA. 1 - Reset the Class 1 CHA.
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. 0 - Don't reset the Class 2 CHA. 1 - Reset the Class 2 CHA.
27 C1D	Clear Class 1 Done Interrupt. Writing a 1 to this bit clears the Class 1 done interrupt. 0 - Don't clear the Class 1 done interrrupt. 1 - Clear the Class 1 done interrrupt.
26 C2D	Clear Class 2 Done Interrupt. Writing a 1 to this bit clears the Class 2 done interrupt. 0 - Don't clear the Class 2 done interrrupt. 1 - Clear the Class 2 done interrrupt.
25 CDS	Clear Descriptor Sharing signal. Writing a 1 to this bit clears the shared_descriptor signal in DECO. This signal tells DECO, and the protocols, whether this descriptor was shared from a previous run. If CDS is set via LOAD IMM to the Clear Written register the fact that this descriptor was shared will be forgotten and the descriptor will behave thereafter as if it was not shared. This is important in protocols where the protocol expects a "decrypt" key but an "encrypt" key is provided. This may occur when using RJD to re-key a flow. Note that writing 1 to this bit when the DECO/CCB is under direct software control will not clear sharing, but that is unimportant because sharing is not possible when the DECO is under direct software control. 0 - Don't clear the shared descriptor signal. 1 - Clear the shared descriptor signal.
24-23 —	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. 0 - Don't clear the Class 2 Key Register. 1 - Clear the Class 2 Key Register.
21 C2C	Clear the Class 2 Context Register. Writing a one to this bit causes the Class 2 Context Register to be cleared. 0 - Don't clear the Class 2 Context Register.

Table continues on the next page...

Field	Description
	1 - Clear the Class 2 Context Register.
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. 0 - Don't clear the Class 2 Data Size Register. 1 - Clear the Class 2 Data Size Register.
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. 0 - Don't clear the Class 2 Mode Register. 1 - Clear the Class 2 Mode Register.
15 CPKE	Clear the PKHA E Size Register. Writing a one to this bit causes the PKHA E Size Register to be cleared. 0 - Don't clear the PKHA E Size Register.. 1 - Clear the PKHA E Size Register.
14 CPKN	Clear the PKHA N Size Register. Writing a one to this bit causes the PKHA N Size Register to be cleared. 0 - Don't clear the PKHA N Size Register. 1 - Clear the PKHA N Size Register.
13 CPKB	Clear the PKHA B Size Register. Writing a one to this bit causes the PKHA B Size Register to be cleared. 0 - Don't clear the PKHA B Size Register. 1 - Clear the PKHA B Size Register.
12 CPKA	Clear the PKHA A Size Register. Writing a one to this bit causes the PKHA A Size Register to be cleared. 0 - Don't clear the PKHA A Size Register. 1 - Clear the PKHA A Size Register.
11-7 —	Reserved
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. 0 - Don't clear the Class 1 Key Register. 1 - Clear the Class 1 Key Register.
5 C1C	Clear the Class 1 Context Register. Writing a one to this bit causes the Class 1 Context Register to be cleared. 0 - Don't clear the Class 1 Context Register. 1 - Clear the Class 1 Context Register.
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. 0 - Don't clear the Class 1 ICV Size Register. 1 - Clear the Class 1 ICV Size Register.

Table continues on the next page...

Field	Description
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. 0 - Don't clear the Class 1 Data Size Register. 1 - Clear the Class 1 Data Size Register.
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. 0 - Don't clear the Class 1 Mode Register. 1 - Clear the Class 1 Mode Register.

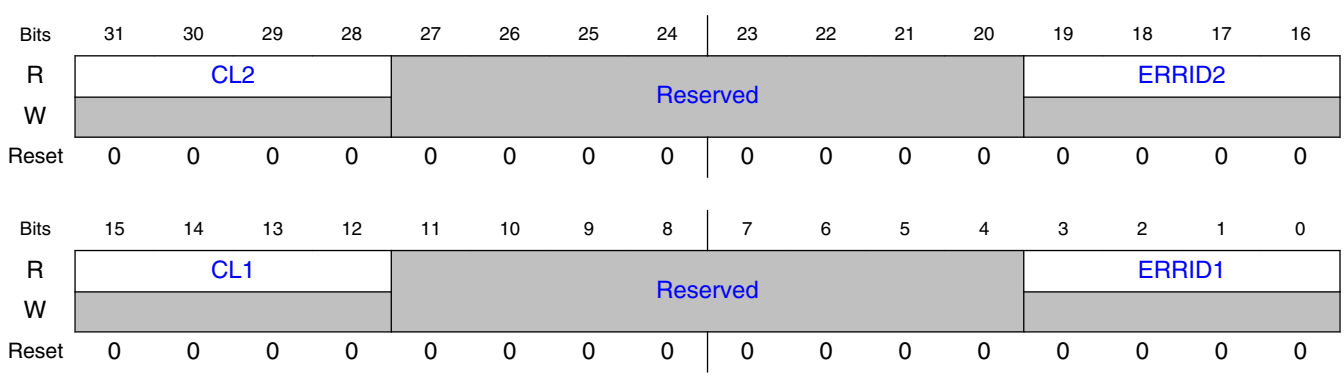
10.13.171 CCB 0 Status and Error Register, most-significant half (C0CSTA_MS)

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

10.13.171.1 Offset

Register	Offset	Description
C0CSTA_MS	8048h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.171.2 Diagram



10.13.171.3 Fields

Field	Description
31-28 CL2	Class 2 Algorithms. The Class 2 Algorithms bits indicate which algorithm is asserting an error. Others reserved. 0100 - MD5, SHA-1, SHA-224, SHA-256
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 Others Reserved 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
15-12 CL1	Class 1 algorithms. The Class 1 algorithms field indicates which algorithm is asserting an error. Others reserved 0001 - AES 0010 - DES 0011 - ARC4 0101 - RNG 1000 - Public Key
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. Others reserved. 0001 - Mode Error 0010 - Data Size Error, including PKHA N Memory Size Error 0011 - Key Size Error, including PKHA E Memory Size Error 0100 - PKHA A Memory Size Error 0101 - PKHA B Memory Size Error 0110 - Data Arrived out of Sequence Error 0111 - PKHA Divide by Zero Error 1000 - PKHA Modulus Even Error

Field	Description
	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 provided, 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

10.13.172 CCB 0 Status and Error Register, least-significant half (C0CSTA_LS)

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

10.13.172.1 Offset

Register	Offset	Description
C0CSTA_LS	804Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.172.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved	PIZ	GCD	PRM	Reserved						SE I	PE I	Reserved		SD I	PDI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RNB	Reserved	MB	P B	Reserved	Reserved	RCB	DB	AB	Reserved
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.172.3 Fields

Field	Description
31 —	Reserved
30 PIZ	Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity. 0 - The result of a Public Key operation is not zero. 1 - The result of a Public Key operation is zero.
29 GCD	GCD is One. The greatest common divisor of two numbers is one (that is, the two numbers are relatively prime). 0 - The greatest common divisor of two numbers is NOT one. 1 - The greatest common divisor of two numbers is one.
28 PRM	Public Key is Prime. The given number is probably prime (that is, it passes the Miller-Rabin primality test). 0 - The given number is NOT prime. 1 - The given number is probably prime.
27-22 —	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 - No Error 1 - Error Interrupt
19-18 —	Reserved
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 —	Reserved
14 —	Reserved
13 —	Reserved
12	Reserved

Table continues on the next page...

Field	Description
—	
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	MDHA Busy. This bit indicates that the MDHA is busy. The CHA can either be busy processing data or resetting. 0 - MDHA Idle 1 - MDHA Busy
6 PB	PKHA Busy. This bit indicates that the Public Key Hardware Accelerator is busy. The CHA can either be busy processing data or resetting. 0 - PKHA Idle 1 - PKHA Busy
5 —	Reserved
4 —	Reserved
3 RCB	AFHA Busy. This bit indicates that the ARC4 Hardware Accelerator is busy. The CHA can either be busy processing data or resetting. 0 - AFHA Idle 1 - AFHA Busy
2 DB	DESA Busy. This bit indicates that the DES Accelerator is busy. The CHA can either be busy processing data or resetting. 0 - DESA Idle 1 - DESA Busy
1 AB	AESA Busy. This bit indicates that the AES Accelerator is busy. The CHA can either be busy processing data or resetting. 0 - AESA Idle 1 - AESA Busy
0 —	Reserved

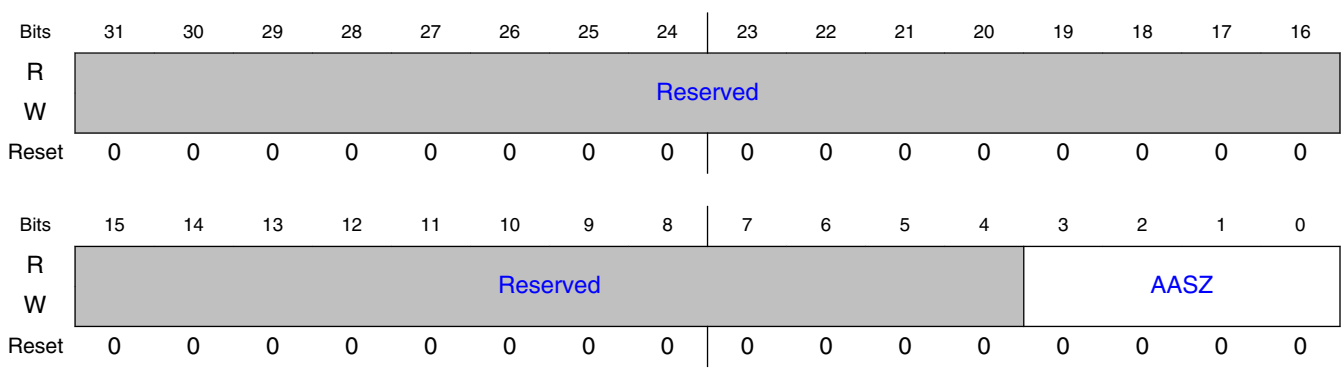
10.13.173 CCB 0 Class 1 AAD Size Register (C0C1AADSZR)

The AAD Size Register is used by AES CHAs 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 register is automatically written by FIFO LOAD commands.

10.13.173.1 Offset

Register	Offset	Description
C0C1AADSZR	805Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.173.2 Diagram



10.13.173.3 Fields

Field	Description
31-4 —	Reserved
3-0 AASZ	AAD size in Bytes, mod 16.

10.13.174 CCB 0 Class 1 IV Size Register (C0C1IVSZR)

The Class 1 IV Size Register tells AES CHAs how much of the last block of IV 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 register is automatically written by FIFO LOAD commands.

10.13.174.1 Offset

Register	Offset	Description
C0C1IVSZR	8064h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.174.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved												IVSZ			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.174.3 Fields

Field	Description
31-4 —	Reserved
3-0 IVSZ	IV size in bytes, mod 16.

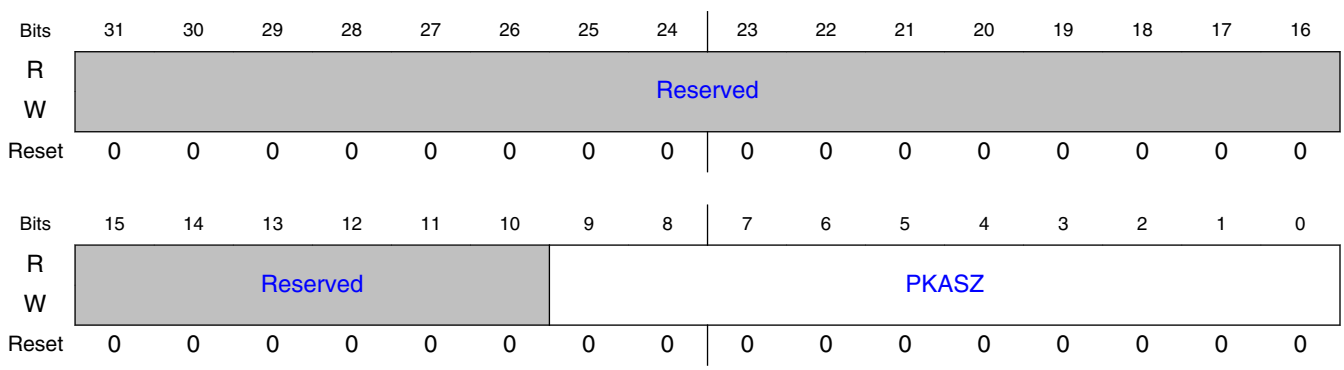
10.13.175 PKHA A Size Register (C0PKASZR)

The PKHA A Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA A Memory. The PKHA A Size Register must be written before the data is written into or read from the PKHA A Memory. This will reserve the PKHA for the current job. The PKHA A Size Register can be automatically written by the MOVE, FIFO LOAD and FIFO STORE commands.

10.13.175.1 Offset

Register	Offset	Description
C0PKASZR	8084h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.175.2 Diagram



10.13.175.3 Fields

Field	Description
31-10 —	Reserved
9-0 PKASZ	PKHA A Memory key size in bytes.

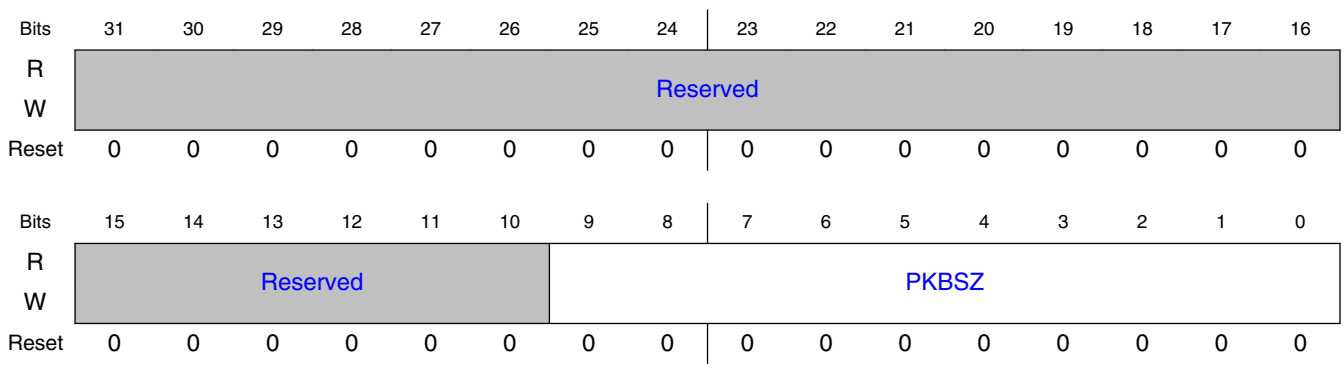
10.13.176 PKHA B Size Register (C0PKBSZR)

The PKHA B Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA B Memory. The PKHA B Size Register must be written before the data is written into or read from the PKHA B Memory. This will reserve the PKHA for the current job. The PKHA B Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

10.13.176.1 Offset

Register	Offset	Description
C0PKBSZR	808Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.176.2 Diagram



10.13.176.3 Fields

Field	Description
31-10 —	Reserved
9-0 PKBSZ	PKHA B Memory key size in bytes.

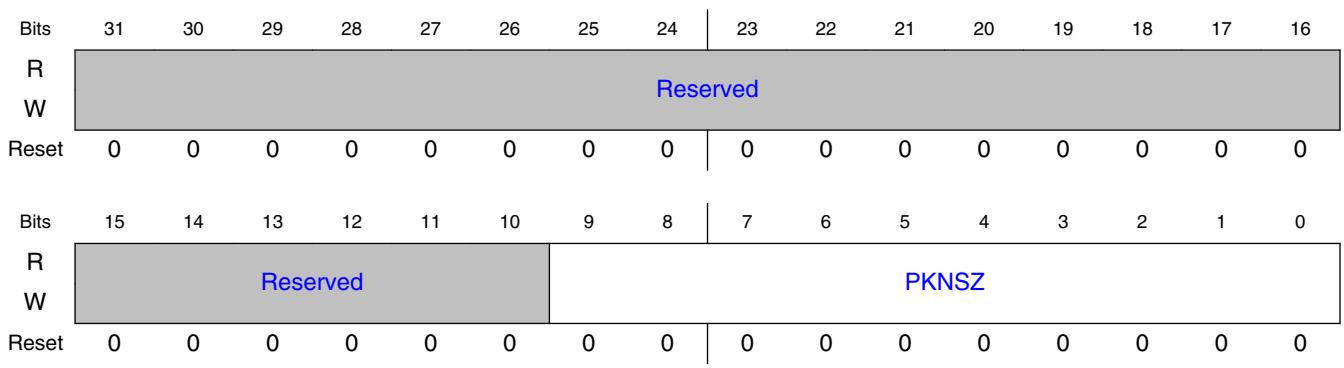
10.13.177 PKHA N Size Register (C0PKNSZR)

The PKHA N Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA N Memory. The PKHA N Size Register must be written before the data is written into or read from the PKHA N Memory. This will reserve the PKHA for the current job. The PKHA N Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

10.13.177.1 Offset

Register	Offset	Description
C0PKNSZR	8094h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.177.2 Diagram



10.13.177.3 Fields

Field	Description
31-10 —	Reserved
9-0 PKNSZ	PKHA N Memory key size in bytes.

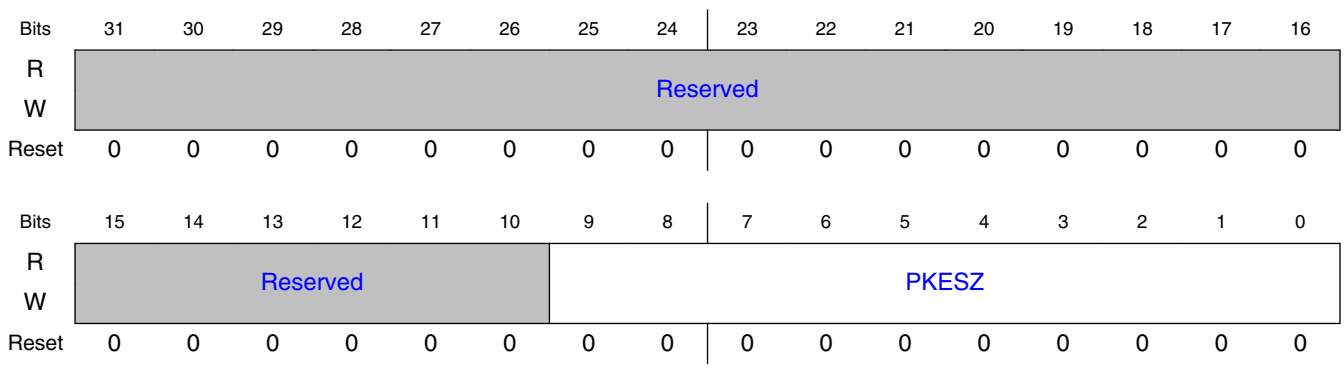
10.13.178 PKHA E Size Register (C0PKESZR)

The PKHA E Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA E Memory. The PKHA E Size Register must be written before the data is written into or read from the PKHA E Memory. This will reserve the PKHA for the current job. The PKHA E Size Register is automatically written by the KEY Command.

10.13.178.1 Offset

Register	Offset	Description
C0PKESZR	809Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.178.2 Diagram



10.13.178.3 Fields

Field	Description
31-10 —	Reserved
9-0 PKESZ	PKHA E Memory key size in bytes.

10.13.179 CCB 0 Class 1 Context Register Word a (C0C1CTXR0 - C0C1CTXR15)

The Class 1 Context Register holds the context for the Class 1 CHAs. 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 1 Context Register is accessible only as full-word reads or writes to sixteen 32-bit registers. The MSB is located at offset 0100h with respect to the register page. 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:

- AES ECB: Section AES ECB mode use of the Context Register
- AES CBC, CBC_CS2, OFB and CFB128: Section AES CBC, OFB, and CFB128 modes use of the Context Register
- AES CTR: Section AES CTR mode use of the Context Register
- AES XCBC_MAC, CMAC: Section AES XCBC-MAC and CMAC Modes use of the Context Register
- AES CCM: Section AES CCM mode use of the Context Register
- AES GCM: Section AES GCM mode use of the Mode Register
- ARC4: Section AFHA use of the Context Register
- DES: Section DESA Context Register
- Random Numbers: Section RNG use of the Context Register
- Triple DES: Section DESA Context Register

10.13.179.1 Offset

For a = 0 to 15:

Register	Offset	Description
C0C1CTXRa	8100h + (a × 4h)	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.179.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	C1CTX															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	C1CTX															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.179.3 Fields

Field	Description
31-0 C1CTX	Class 1 Context.

10.13.180 CCB 0 Class 1 Key Registers Word a (C0C1KR0 - C0C1 KR7)

The Class 1 Key Register normally holds the left-aligned key for the Class 1 CHAs. The MSB is in offset 200h. The Class 1 Key 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. Although the Class 1 Key Register is only 32 bytes long, via the KEY command it is possible to load a key larger than 32 bytes. In this case part of the Class 1 Context Register is used as an "Extended Key Register". The first 32 bytes are loaded into the Class 1 Key Register, then, starting with the most-significant end, as many 8-byte chunks of the Class 1 Context Register as required are allocated to the Extended Key Register and are used to hold the remaining key bytes. The Extended Key Register bytes cannot be overwritten and will return 0 when read. The remaining bytes are still available for context data. Clearing the Class 1 Key Register will also clear the Extended Key

Register bytes in the Class 1 Context Register. The other bytes in the Class 1 Context Register will not be cleared. Note that clearing the Class 1 Context Register will not clear the Extended Key Register bytes.

The Class 1 Key Register can be written via a MOVE Command, a MATH Command, a LOAD Command or a KEY Command. Before the value in the Class 1 Key Register can be used in a cryptographic operation, the size of the key must be written into the Class 1 Key Size Register. Once the Class 1 Key Size Register has been written, the Class 1 Key Register cannot be written again until the Class 1 Key Size Register has been cleared. Writing the Class 1 Key Register via a KEY Command automatically writes the Class 1 Key Size Register, but if the Class 1 Key Register is written using a MOVE, MATH or LOAD Command the Class 1 Key Size Register must be written via a separate command after the Class 1 Key Register has been written. But until the Class 1 Key Size Register has been written the Class 1 Key Register remains writable via STORE/SEQ STORE, MATH or MOVE commands and readable via LOAD/SEQ LOAD, MATH or MOVE commands. If the Class 1 Key Size Register and the Class 1 Key Register have been cleared via the Clear Written Register, the Class 1 Key Register becomes writable and readable again. This allows the Class 1 Key Register to be used for temporary storage if it is not currently needed to hold a cryptographic key.

Even when the Class 1 Key Register holds a key (i.e. the Class 1 Key Size Register has been written) it may still be possible to store the key in memory in encrypted form. The FIFO STORE Command can be used to store an encrypted copy of this key (i.e. a Black Key), unless storing the key has been prohibited via the NWB bit in the KEY Command. The encrypted key can later be loaded into the Class 1 Key Register via the KEY Command by setting the ENC bit to indicate that this is a Black (i.e. encrypted) Key. The Black Key will automatically be decrypted before it is loaded into the Class 1 Key Register. A Black Key can be loaded as long as the Key Encryption Key (KEK) has not been changed (as a consequence of a security violation or a POR). Note that the Class 1 Key register is cleared when any key (including Class 2 Keys) is encrypted or decrypted, so if a Black Key is to be loaded into or stored from the Class 2 Key Register, that must be done prior to loading a key into the Class 1 Key Register. Similarly, if a key is to be stored from the Class 1 Key Register as a Black Key and also used in a cryptographic operation, the cryptographic operation should be performed first, or the key will have to be loaded a second time.

10.13.180.1 Offset

For a = 0 to 7:

Register	Offset	Description
C0C1KRa	8200h + (a × 4h)	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.180.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	C1KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	C1KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.180.3 Fields

Field	Description
31-0 C1KEY	Class 1 Key.

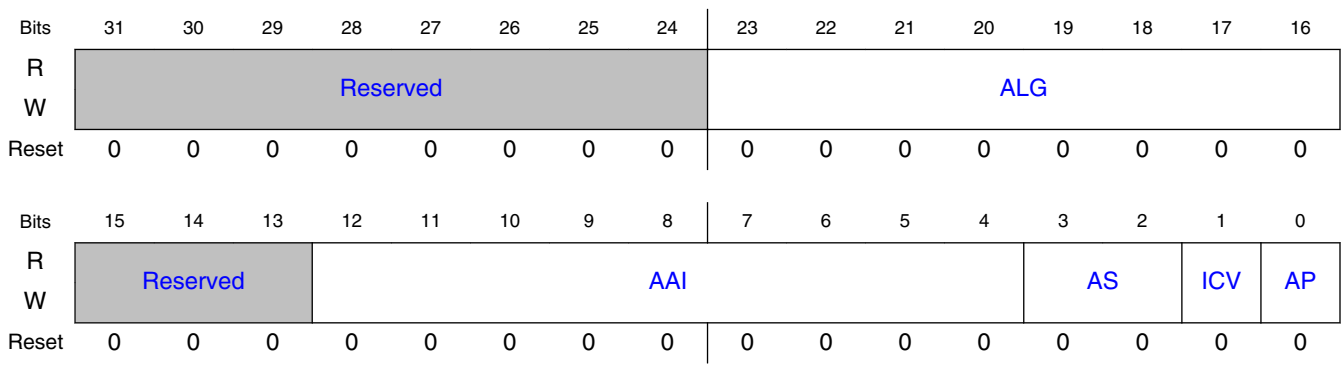
10.13.181 CCB 0 Class 2 Mode Register (C0C2MR)

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.

10.13.181.1 Offset

Register	Offset	Description
C0C2MR	8404h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.181.2 Diagram



10.13.181.3 Fields

Field	Description
31-24 —	Reserved
23-16 ALG	Algorithm. This field specifies which algorithm has been requested for an OPERATION command. 01000000 - MD5 01000001 - SHA-1 01000010 - SHA-224 01000011 - SHA-256
15-13 —	Reserved
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.

Value	Description	Valid with ALG
000h	Hash without key	MD5, SHA-*
001h	HMAC using a non-derived key	MD5, SHA-*
002h	SMAC	MD5, SHA-1

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Value</th><th>Description</th><th>Valid with ALG</th></tr>
<tr><td>004h</td><td>HMAC using a derived key</td><td>MD5, SHA-*</td></tr>
<tr><td>Others</td><td>Reserved</td><td></td></tr>
</table>
</td></tr>
<tr><td>3-2
AS</td><td>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.</td></tr>
<tr><td>1
ICV</td><td>ICV Checking. If this bit is set, the calculated ICV will be compared against a received ICV. This bit will be ignored by algorithms that do not support ICV checking.
0 - Don't compare the calculated ICV against a received ICV.
1 - Compare the calculated ICV against a received ICV.</td></tr>
<tr><td>0
AP</td><td>Authenticate / Protect. Used by the Performance Counter to determine which count register to update.
0 - Authenticate
1 - Protect</td></tr>
</table>

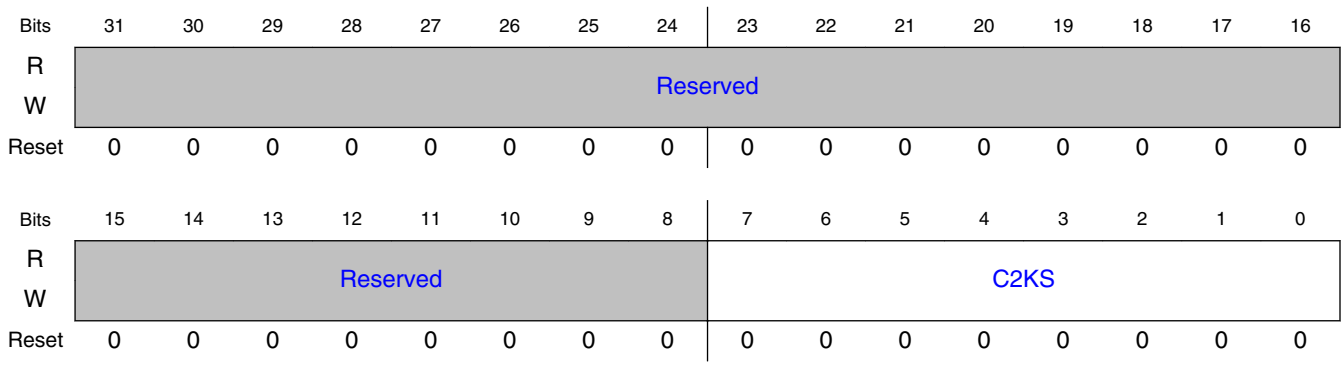
10.13.182 CCB 0 Class 2 Key Size Register (C0C2KSR)

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 Class 2 Key Size Register is automatically written by the Key Command. This register is cleared when Trusted Descriptors are checked or re-signed.

10.13.182.1 Offset

Register	Offset	Description
C0C2KSR	840Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.182.2 Diagram



10.13.182.3 Fields

Field	Description
31-8 —	Reserved
7-0 C2KS	Class 2 key size in bytes.

10.13.183 CCB 0 Class 2 Data Size Register (C0C2DSR)

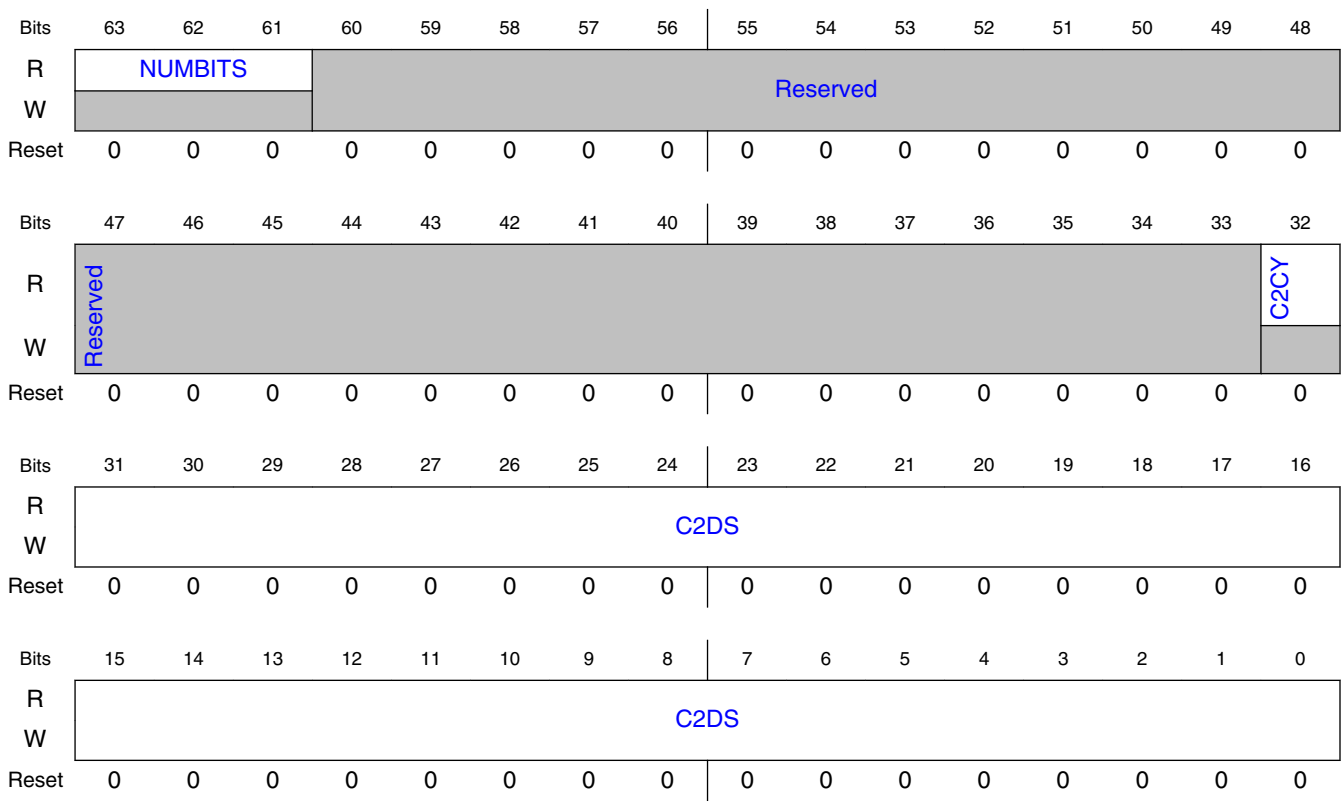
The Class 2 Data Size Register is used to tell the Class 2 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C2CY and C2DS fields to form a data size that is measured in bits. Note that writing to the C2DS 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.

10.13.183.1 Offset

Register	Offset	Description
C0C2DSR	8410h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.183.2 Diagram



10.13.183.3 Fields

Field	Description
63-61 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.
60-33 —	Reserved
32 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. 0 - A write to the Class 2 Data Size Register did not cause a carry. 1 - A write to the Class 2 Data Size Register caused a carry.
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.

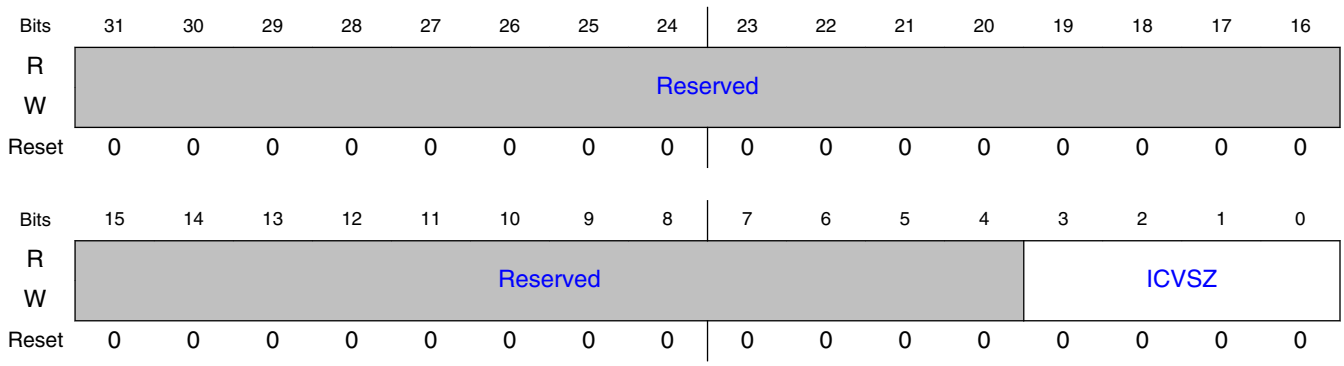
10.13.184 CCB 0 Class 2 ICV Size Register (C0C2ICVSZR)

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, SHA-384, and MD5). For AES Class 2 operations the Class 2 ICV Size Register indicates the size of the ICV. Writing to this register causes the written value to be added to the previous value in the register. This register is automatically written by FIFO LOAD commands. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

10.13.184.1 Offset

Register	Offset	Description
C0C2ICVSZR	841Ch	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.184.2 Diagram



10.13.184.3 Fields

Field	Description
31-4 —	Reserved
3-0 ICVSZ	Class 2 ICV size (mod 8) in bytes. For MDHA, writing 0 to this field will be interpreted as an ICV size of 8 bytes. For AESA, writing 0, 1, 2 or 3 to this field will be interpreted as an ICV size of 16 bytes.

10.13.185 CCB 0 Class 2 Context Register Word a (C0C2CTXR0 - C0C2CTXR9)

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 located at offset 500h with respect to the register page. 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.

- MD5: Section MDHA use of the Context Register
- SHA-*: Section MDHA use of the Context Register

10.13.185.1 Offset

For a = 0 to 9:

Register	Offset	Description
C0C2CTXRa	8500h + (a × 4h)	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.185.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	C2CTXR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	C2CTXR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.185.3 Fields

Field	Description
31-0 C2CTXR	Class 2 Context.

10.13.186 CCB 0 Class 2 Key Register Word a (C0C2KEYR0 - C0C2KEYR15)

The Class 2 Key Register holds the key for the Class 2 CHAs. For non-derived HMAC keys, the key is left-aligned in the C2KEYR. Derived HMAC keys consist of two derivations from the original non-derived HMAC key: For the first derivation the original HMAC key is XORed with IPAD (a constant byte 36h repeated to fill a block) and then processed with the underlying hash function. The second derivation consists of the HMAC key getting XORed with OPAD (a constant byte 5Ch repeated to fill a block) and

then processed with the underlying hash function. Note that the size of the derived HMAC key is twice the size of of the Message Digest registers for the underlying hash function. The IPAD-half of the derived key is left-aligned within C2KEYR, while the OPAD-half of the derived key 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 located at offset 600h with respect to the start of the register page. 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.

10.13.186.1 Offset

For a = 0 to 15:

Register	Offset	Description
C0C2KEYRa	8600h + (a × 4h)	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.186.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	C2KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	C2KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.186.3 Fields

Field	Description
31-0 C2KEY	Class 2 Key.

10.13.187 CCB 0 FIFO Status Register (C0FIFOSTA)

The CCB FIFO Status Register is used during debug to facilitate reading the CCB FIFOs. Software must keep track of the data written to the Input Data FIFO (CCB 0 Input Data FIFO (C0IFIFO)), but the data within the Output Data FIFO (CCB 0 Output Data FIFO (C0OFIFO)) can be read out. Both the Class 1 Alignment Block and the Class 2 Alignment Block (see Alignment blocks) draw data from the Input Data FIFO, and both the DMA and the DECO Alignment Block draw data from the Output Data FIFO. Reading the CaFIFOSTA register returns the current heads of the Alignment Block and DMA queues within these two FIFOs. Note that the values in this register will change as descriptors are executed, so the register should be read when the DECO is not actively executing a descriptor.

10.13.187.1 Offset

Register	Offset	Description
C0FIFOSTA	87C0h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.187.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	C1IQHEAD								C2IQHEAD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMAOQHEAD								DECOOQHEAD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.187.3 Fields

Field	Description
31-24 C1IQHEAD	This is the current head of the Class 1 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 1 Alignment Block.
23-16 C2IQHEAD	This is the current head of the Class 2 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 2 Alignment Block.
15-8 DMAOQHEAD	This is the current head of the DMA queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DMA controller.
7-0 DECOOQHEAD	This is the current head of the DECO Alignment Block queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DECO Alignment Block. This is used during "out snooping" operations, i.e. when data is passed first through a Class 1 CHA and the results pushed into the OFIFO, and from there the results are sent through a Class 2 CHA.

10.13.188 CCB 0 iNformation FIFO When STYPE != 10b (C0NFIFO)

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 Alignment blocks). 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 that does not automatically generate an info FIFO entry, the user is responsible for writing to the iNformation FIFO. See LOAD Command destination codes 78h and 7Ah in LOAD commands.)

A single address is used to write to the iNformation FIFO. The format of non-padding iNformation FIFO entries (STYPE != 10b) is shown below. The format of padding iNformation FIFO entries (STYPE == 10b) is shown in CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2).

10.13.188.1 Offset

Register	Offset	Description
C0NFIFO	87D0h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.188.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	DEST		LC2	LC1	FC2	FC1	STYPE		DTYPE				BND	PTYPE		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	OC	AST	Reserved		DL											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.188.3 Fields

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 DTYPE == Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPE 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. 0 - This is not the last Class 2 data. 1 - This is the last Class 2 data.
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. 0 - This is not the last Class 1 data.

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>1 - This is the last Class 1 data.</td></tr>
<tr><td>27
FC2</td><td>Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted.
This bit can be set only 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.
0 - Don't flush Class 2 data.
1 - Flush Class 2 data.</td></tr>
<tr><td>26
FC1</td><td>Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.
0 - Don't flush Class 1 data.
1 - Flush Class 1 data.</td></tr>
<tr><td>25-24
STYPE</td><td>Source Type. This field defines the source of the data for the Alignment Block(s). (This is the register format description when STYPE != 10b. The register uses a different format when STYPE == 10b. See CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2).) For STYPE != 10b, there are two interpretations of the STYPE field, depending on the setting of the AST bit:
<table><tr><th>AST=0</th><th>AST=1</th></tr><tr><td>STYPE = 00b : Input Data FIFO</td><td>STYPE = 00b : Auxiliary Data FIFO [-1]</td></tr><tr><td>STYPE = 01b : Output Data FIFO</td><td>STYPE = 01b : Output Data FIFO Synchronize Pointers [-1]</td></tr><tr><td colspan="2">STYPE = 10b : Padding Block. The register format is different for this STYPE. See CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2).</td></tr><tr><td>STYPE = 11b : Out snooping [-1]</td><td>STYPE = 11b : Outsnooping from Auxiliary Data FIFO [-1]</td></tr></table>
1. 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=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should 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 may hang.
2. The output FIFO maintains two pointers: one for data being pulled by alignment blocks and one for the DMAs pulling data. When STYPE=01b and AST=0, the alignment block pointer moves independently from the DMA pointer. In such cases, if one pointer laps the other when reading from the output FIFO, reading from that pointer must stop until the lapped pointer moves. This prevents data for the other pointer from being lost. If no DMA transactions are scheduled and the alignment block wants to consume more data than will fit in the output FIFO, the descriptor will hang. In such situations, instead use STYPE=01b and AST=1. This combination forces the two pointers to remain synchronized, allowing the alignment block reads to drain the output FIFO rather than leaving the data in the FIFO for the DMA to read.
3. 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.
4. This case is similar to the case of STYPE=11b and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.</td></tr>
<tr><td>23-20
DTYPE</td><td>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. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.
<table><tr><td>DTYPE</td><td>Data type for PKHA</td><td>Data type for DECO and other CHA(s)</td></tr></table></td></tr>
</table>

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>0h</td><td>PKHA A0</td><td>S-Box (AFHA)</td></tr>
<tr><td>1h</td><td>PKHA A1</td><td>AAD (AES GCM)</td></tr>
<tr><td>2h</td><td>PKHA A2</td><td>IV for (AES GCM)</td></tr>
<tr><td>3h</td><td>PKHA A3</td><td>SAD (AES)</td></tr>
<tr><td>4h</td><td>PKHA B0</td><td>Reserved</td></tr>
<tr><td>5h</td><td>PKHA B1</td><td>Reserved</td></tr>
<tr><td>6h</td><td>PKHA B2</td><td>Reserved</td></tr>
<tr><td>7h</td><td>PKHA B3</td><td>Reserved</td></tr>
<tr><td>8h</td><td>PKHA N</td><td>Reserved</td></tr>
<tr><td>9h</td><td>PKHA E</td><td>Reserved</td></tr>
<tr><td>Ah</td><td>Reserved</td><td>ICV</td></tr>
<tr><td>Ch</td><td>PKHA A</td><td>Reserved</td></tr>
<tr><td>Dh</td><td>PKHA B</td><td>Reserved</td></tr>
<tr><td>Eh</td><td>Reserved</td><td>DECO Ignore (i.e. Skip)</td></tr>
<tr><td>Fh</td><td>Reserved</td><td>Message Data</td></tr>
<tr><td colspan="3">Other DTYPE values are reserved in this instance of CAAM.</td></tr>
</table>

1. 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=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should 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 may hang.
2. The output FIFO maintains two pointers: one for data being pulled by alignment blocks and one for the DMAs pulling data. When STYPE=01b and AST=0, the alignment block pointer moves independently from the DMA pointer. In such cases, if one pointer laps the other when reading from the output FIFO, reading from that pointer must stop until the lapped pointer moves. This prevents data for the other pointer from being lost. If no DMA transactions are scheduled and the alignment block wants to consume more data than will fit in the output FIFO, the descriptor will hang. In such situations, instead use STYPE=01b and AST=1. This combination forces the two pointers to remain synchronized, allowing the alignment block reads to drain the output FIFO rather than leaving the data in the FIFO for the DMA to read.
3. 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.
4. This case is similar to the case of STYPE=11b and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.</td></tr>
<tr><td>19
BND</td><td>Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE == 10b (Padding Block).
0 - Don't pad.
1 - Pad to the next 16-byte boundary.</td></tr>
<tr><td>18-16
PTYPE</td><td>Pad Type. This field is ignored if STYPE != 10b (i.e., not Padding Block). See CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2) for information on how PTYPE is used when STYPE == 10b.</td></tr>
<tr><td>15
OC</td><td>OFIFO Continuation - This bit causes the final word to not be popped from the Output Data FIFO.
0 - Allow the final word to be popped from the Output Data FIFO.
1 - Don't pop the final word from the Output Data FIFO.</td></tr>
</table>

Table continues on the next page...

Field	Description
14 AST	Additional Source Types. This bit selects between two meanings of the STYPE field. See the description of the STYPE field.
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.

1. 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=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should 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 may hang.
2. The output FIFO maintains two pointers: one for data being pulled by alignment blocks and one for the DMAs pulling data. When STYPE=01b and AST=0, the alignment block pointer moves independently from the DMA pointer. In such cases, if one pointer laps the other when reading from the output FIFO, reading from that pointer must stop until the lapped pointer moves. This prevents data for the other pointer from being lost. If no DMA transactions are scheduled and the alignment block wants to consume more data than will fit in the output FIFO, the descriptor will hang. In such situations, instead use STYPE=01b and AST=1. This combination forces the two pointers to remain synchronized, allowing the alignment block reads to drain the output FIFO rather than leaving the data in the FIFO for the DMA to read.
3. 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.
4. This case is similar to the case of STYPE=11b and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.

10.13.189 CCB 0 iNformation FIFO When STYPE == 10b (C0NFIFO_2)

The format of padding iNformation FIFO entries (STYPE == 10b) is shown below. The format of non-padding iNformation FIFO entries (STYPE != 10b) is shown in CCB 0 iNformation FIFO When STYPE != 10b (C0NFIFO).

10.13.189.1 Offset

Register	Offset	Description
C0NFIFO_2	87D0h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.189.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	DEST		LC2	LC1	FC2	FC1	STYPE		DTYPE				BND	PTYPE		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		Reserved					Reserved									
W	PR				BM	PS				PL						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.189.3 Fields

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 DTYPE 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 DTYPE 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. 0 - This is not the last Class 2 data. 1 - This is the last Class 2 data.
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. 0 - This is not the last Class 1 data. 1 - This is the last Class 1 data.
27 FC2	Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted. 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. 0 - Don't flush the Class 2 data. 1 - Flush the Class 2 data.
26 FC1	Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block. 0 - Don't flush the Class 1 data.

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>1 - Flush the Class 1 data.</td></tr>
<tr><td>25-24
STYPE</td><td>Source Type. This field defines the source of the data for the Alignment Block(s). This is the register format description when STYPE == 10b (Padding Block). For STYPE != 10b, see CCB 0 iNformation FIFO When STYPE != 10b (C0NFIFO).

* 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.

** 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=78h 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.</td></tr>
<tr><td>23-20
DTYPE</td><td>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. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.

<table>
<tr><th>DTYPE</th><th>Data type for PKHA</th><th>Data type for DECO and other CHA(s)</th></tr>
<tr><td>0h</td><td>PKHA A0</td><td>S-Box for AFHA</td></tr>
<tr><td>1h</td><td>PKHA A1</td><td>AAD for AES GCM</td></tr>
<tr><td>2h</td><td>PKHA A2</td><td>IV for AES GCM</td></tr>
<tr><td>3h</td><td>PKHA A3</td><td>SAD for AES</td></tr>
<tr><td>4h</td><td>PKHA B0</td><td>Reserved</td></tr>
<tr><td>5h</td><td>PKHA B1</td><td>Reserved</td></tr>
<tr><td>6h</td><td>PKHA B2</td><td>Reserved</td></tr>
<tr><td>7h</td><td>PKHA B3</td><td>Reserved</td></tr>
<tr><td>8h</td><td>PKHA N</td><td>Reserved</td></tr>
<tr><td>9h</td><td>PKHA E</td><td>Reserved</td></tr>
<tr><td>Ah</td><td>Reserved</td><td>ICV</td></tr>
<tr><td>Ch</td><td>PKHA A</td><td>Reserved</td></tr>
<tr><td>Dh</td><td>PKHA B</td><td>Reserved</td></tr>
<tr><td>Eh</td><td>Reserved</td><td>DECO Ignore (i.e. Skip)</td></tr>
<tr><td>Fh</td><td>Reserved</td><td>Message Data</td></tr>
<tr><td colspan="3">Other DTYPE values are reserved in this instance of CAAM.</td></tr>
</table>
</td></tr>
<tr><td>19
BND</td><td>Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE = Padding Block.

0 - Don't add boundary padding.

1 - Add boundary padding.</td></tr>
<tr><td>18-16
PTYPE</td><td>Pad Type. This field defines the type of padding that should be performed for N bytes when the STYPE == 10b (Padding Block). This field is ignored if BND == 0 or STYPE != 10b (Padding Block).

000 - All Zero.

001 - Random with nonzero bytes.

010 - Incremented (starting with 01h), followed by a byte containing the value N-1, i.e., if N==1, a single byte is output with value 0h.

011 - Random.</td></tr>
</table>

Table continues on the next page...

Field	Description
	100 - All Zero with last byte containing the number of 0 bytes, i.e., if N==1, a single byte is output with value 0h. 101 - Random with nonzero bytes with last byte 0. 110 - N bytes of padding all containing the value N-1. 111 - Random with nonzero bytes, with the last byte containing the value N-1.
15 PR	Prediction Resistance - If PTYPE specifies random data, setting PR=1 causes the RNG to supply random data for prediction resistance (i.e. reseeds the PRNG from the TRNG). 0 - No prediction resistance. 1 - Prediction resistance.
14-12 —	Reserved
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. 0 - When padding, pad to power-of-2 boundary. 1 - When padding, pad to power-of-2 boundary minus 1 byte.
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. 0 - C2 CHA snoops pad data from padding block. 1 - C2 CHA snoops pad data from OFIFO.
9-7 —	Reserved
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.

10.13.190 CCB 0 Input Data FIFO (C0IFIFO)

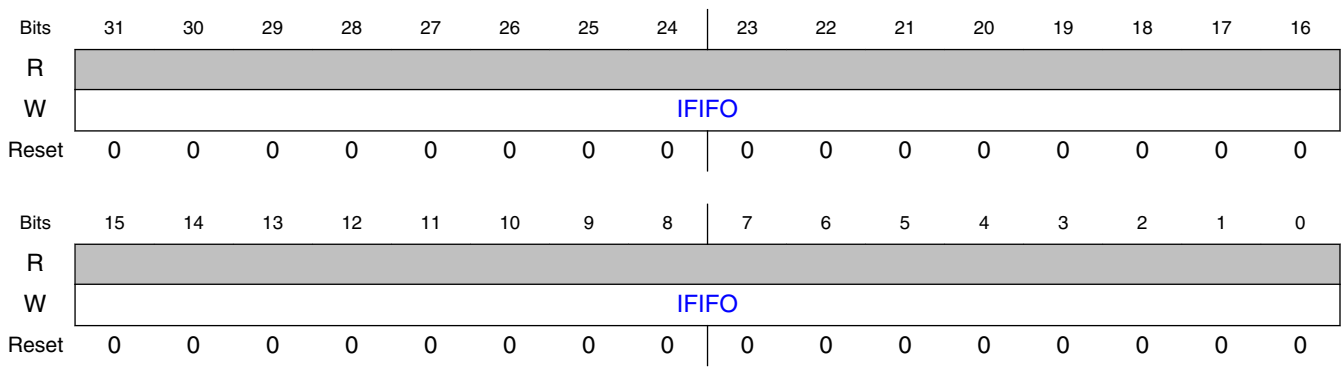
Data to be processed by the various CHAs is first pushed into the Input Data FIFO. Note that although the Input Data FIFO is 64-bits wide, a single 32-bit wide location is used to write data to the IFIFO. All data written to this location via the IP bus should be in big endian format. Like the other DECO/CCB registers, 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 (see “Input Data FIFO Nibble Shift Register”, value 76h, in LOAD commands). The IFIFO is eight 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.

10.13.190.1 Offset

Register	Offset	Description
C0IFIFO	87E0h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.190.2 Diagram



10.13.190.3 Fields

Field	Description
31-0 IFIFO	Input Data FIFO.

10.13.191 CCB 0 Output Data FIFO (C0OFIFO)

Data that is output from the various CHAs is pushed into the Output Data FIFO. The OFIFO is eight 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 (see Output Data FIFO Nibble Shift Register, value 77h, in LOAD commands).

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 or various PKHA registers 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 BASE+x7F4, and the least significant half is read from address BASE+x7F0. All data read from the OFIFO is little endian.

10.13.191.1 Offset

Register	Offset	Description
C0OFIFO	87F0h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.191.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	OFIFO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	OFIFO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OFIFO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OFIFO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.191.3 Fields

Field	Description
63-0 OFIFO	Output FIFO

10.13.192 DECO0 Job Queue Control Register, most-significant half (D0JQCR_MS)

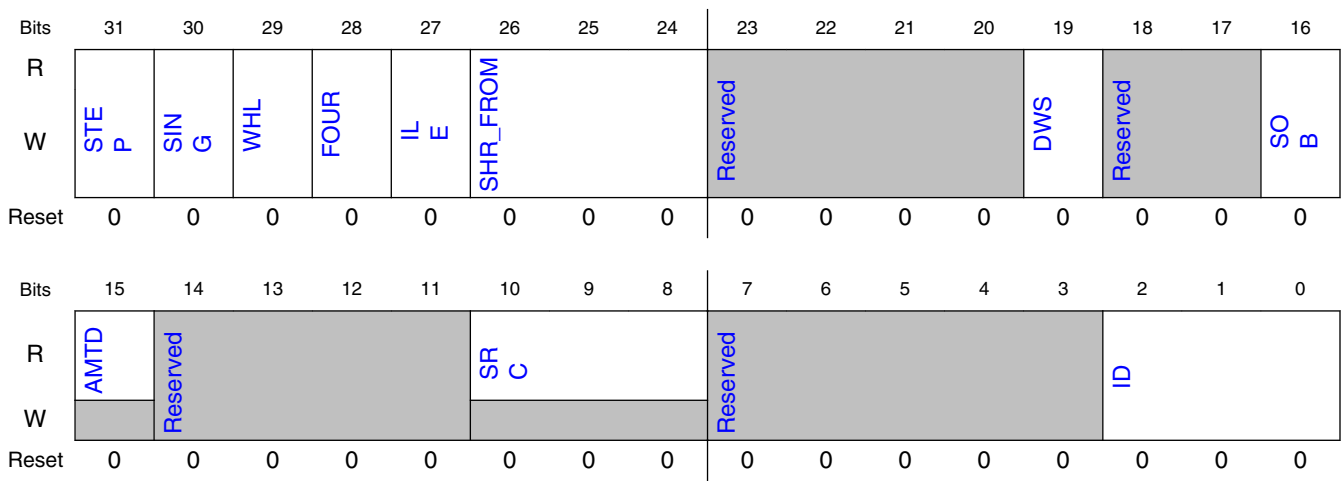
This register tells the Descriptor Controller about the current Descriptor. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see Register-based service interface), this register can be read or written from the IP Register 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 and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

10.13.192.1 Offset

Register	Offset	Description
D0JQCR_MS	8800h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.192.2 Diagram



10.13.192.3 Fields

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. 0 - DECO has not been told to execute the next command in the descriptor. 1 - DECO has been told to execute the next command in the descriptor.
30 SING	Single Step Mode. This tells DECO to execute this descriptor, including jumps to non-local destinations, in single step mode. Only used by the processor that has control of DECO. 0 - Do not tell DECO to execute the descriptor in single-step mode. 1 - Tell DECO to execute the descriptor in single-step mode.

Table continues on the next page...

Field	Description
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. 0 - DECO has not been given the whole descriptor. 1 - DECO has been given the whole descriptor.
28 FOUR	Four Words. job queue controller is passing at least 4 words of the Descriptor to DECO. 0 - DECO has not been given at least four words of the descriptor. 1 - DECO has been given at least four words of the descriptor.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. Byte-swapping is controlled when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: • Input Data FIFO • Output Data FIFO • Auxiliary Data FIFO • Class 1 Context register • Class 2 Context register • Class 1 Key register • Class 2 Key register 0 - No byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer. 1 - Byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer.
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).
23-20 —	Reserved
19 DWS	Double word swap. Causes/allows dword swapping of addresses, and MOVE and MATH immediate values. 0 - Double Word Swap is NOT set. 1 - Double Word Swap is set.
18-17 —	Reserved
16 SOB	Shared Descriptor burst. If set, the whole shared descriptor was passed to DECO with the Job Descriptor. When descriptors are executed under direct software control, this bit simply indicates that the Shared Descriptor has been loaded. 0 - Shared Descriptor has NOT been loaded. 1 - Shared Descriptor HAS been loaded.
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 JRaDID Register. 0 - The Allowed Make Trusted Descriptor bit was NOT set. 1 - The Allowed Make Trusted Descriptor bit was set.
14-11 —	Reserved

Table continues on the next page...

Field	Description
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS) and endian configuration bits) the DMA should use for bus 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). 000 - Job Ring 0 001 - Job Ring 1 010 - Job Ring 2 011 - Reserved 100 - RTIC 101 - Reserved 110 - Reserved 111 - Reserved
7-3 —	Reserved
2-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.

10.13.193 DECO0 Job Queue Control Register, least-significant half (D0JQCR_LS)

This register tells the Descriptor Controller about the current Descriptor. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see Register-based service interface), this register can be read or written from the IP Register 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 and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

10.13.193.1 Offset

Register	Offset	Description
D0JQCR_LS	8804h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.193.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.193.3 Fields

Field	Description
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.

10.13.194 DECO0 Descriptor Address Register (D0DAR)

This DECO register holds the address of the currently executing Job Descriptor. When using DECO in single-step mode (see Register-based service interface), this register must be written prior to the Job Queue Control Register.

10.13.194.1 Offset

Register	Offset	Description
D0DAR	8808h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.194.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.194.3 Fields

Field	Description
63-32 —	Reserved
31-0 DPTR	Descriptor Pointer. Memory address of the Descriptor. Needed for write-back purposes.

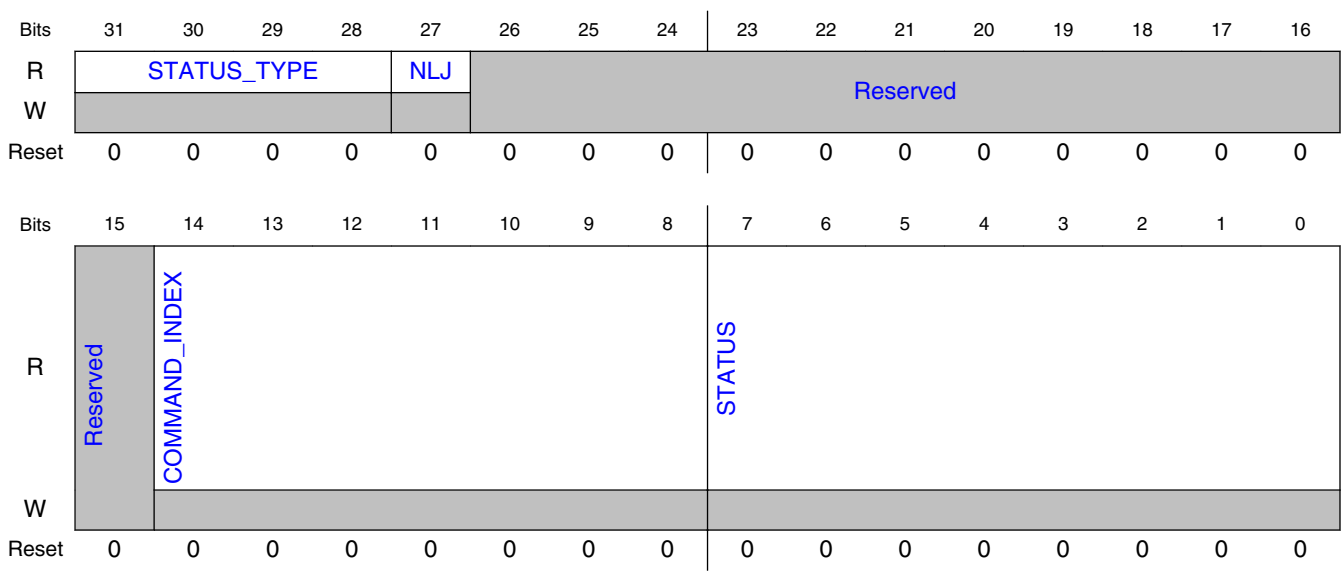
10.13.195 DECO0 Operation Status Register, most-significant half (D0OPSTA_MS)

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 Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

10.13.195.1 Offset

Register	Offset	Description
D0OPSTA_MS	8810h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.195.2 Diagram



10.13.195.3 Fields

Field	Description
31-28 STATUS_TYPE	Status Type. The type of status that is being reported for the just-completed command, defined as follows: 0000 - no error 0001 - DMA error 0010 - CCB error

Table continues on the next page...

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>0011 - Jump Halt User Status
0100 - DECO error
0101, 0110 - Reserved
0111 - Jump Halt Condition Code</td></tr>
<tr><td>27
NLJ</td><td>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.
0 - The original job descriptor running in this DECO has not caused another job descriptor to be executed.
1 - The original job descriptor running in this DECO has caused another job descriptor to be executed.</td></tr>
<tr><td>26-15
—</td><td>Reserved</td></tr>
<tr><td>14-8
COMMAND_INDEX</td><td>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).</td></tr>
<tr><td>7-0
STATUS</td><td>If ERRTYP indicates no error, this field contains PKHA/Math Status, as defined below. If there was an error, this field contains Error Status, defined as in the Job Ring Output Status Register DESC ERROR field (Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR2)).
<table>
<tr><td>7
PIZ</td><td>Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity.</td></tr>
<tr><td>6
GCD</td><td>GCD is One. The greatest common divisor of two numbers (i.e., the two numbers are relatively prime)</td></tr>
<tr><td>5
PRM</td><td>Public Key is Prime. The given number is probably prime (i.e., it passes the Miller-Rabin primality test)</td></tr>
<tr><td>4</td><td>Reserved</td></tr>
<tr><td>3
MN</td><td>Math N. The result is negative. Can only be set by add and subtract functions, 0 otherwise</td></tr>
<tr><td>2
MZ</td><td>Math Z. The result of a math operation is zero.</td></tr>
<tr><td>1
MC</td><td>Math C. The math operation resulted in a carry or borrow.</td></tr>
<tr><td>0
MNV</td><td>Math NV. Used for signed compares. This is a combination of the sign and overflow bits (i.e., Math N XOR Math C)</td></tr>
</table>
</td></tr>
</table>

10.13.196 DECO0 Operation Status Register, least-significant half (D0OPSTA_LS)

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 Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

10.13.196.1 Offset

Register	Offset	Description
D0OPSTA_LS	8814h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.196.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OUT_CT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OUT_CT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.196.3 Fields

Field	Description
31-0 OUT_CT	Output Count. Number of bytes written to sequential out pointer.

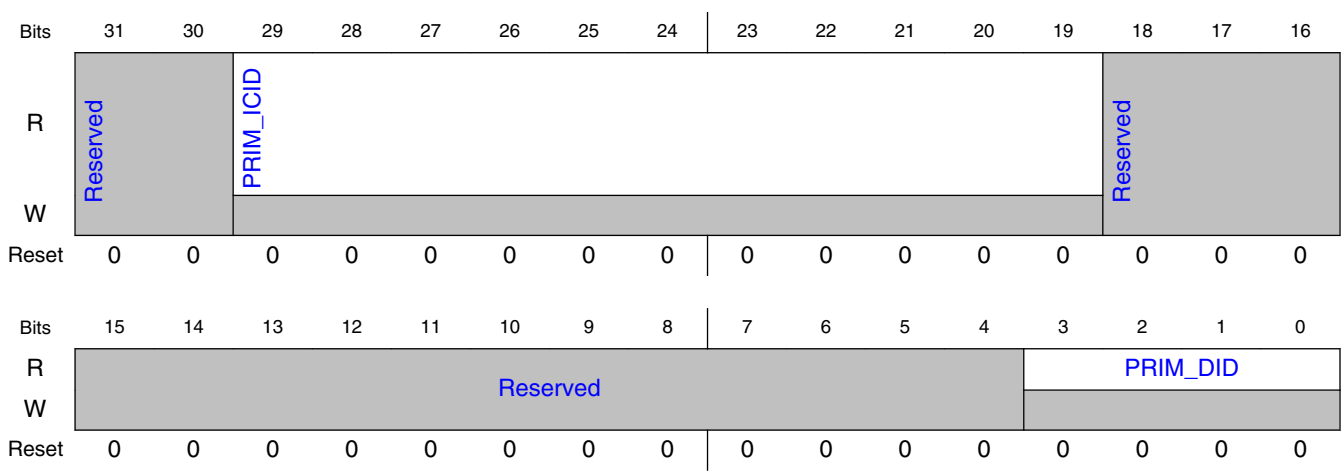
10.13.197 DECO0 Primary DID Status Register (D0PDIDSR)

This register shows the DIDs and ICIDs available to the DECO for processing the current Descriptor. This register is written by the job queue controller.

10.13.197.1 Offset

Register	Offset	Description
D0PDIDSR	8820h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.197.2 Diagram



10.13.197.3 Fields

Field	Description
31-30 —	Reserved
29-19 PRIM_ICID	DECO Primary ICID. When JRaDID[USE_OUT] = 0, this is the ICID that is asserted during all DMA transactions associated with this Job Ring. When JRaDID[USE_OUT] = 0, this is the ICID value is asserted for job termination status writes and job descriptor update writes.
18-4 —	Reserved

Table continues on the next page...

Field	Description
3-0 PRIM_DID	DECO Primary DID. When JRaDID[USE_OUT] = 0, this is the DID that is asserted during all DMA transactions associated with this Job Ring. When JRaDID[USE_OUT] = 0, this is the DID value is asserted for job termination status writes and job descriptor update writes.

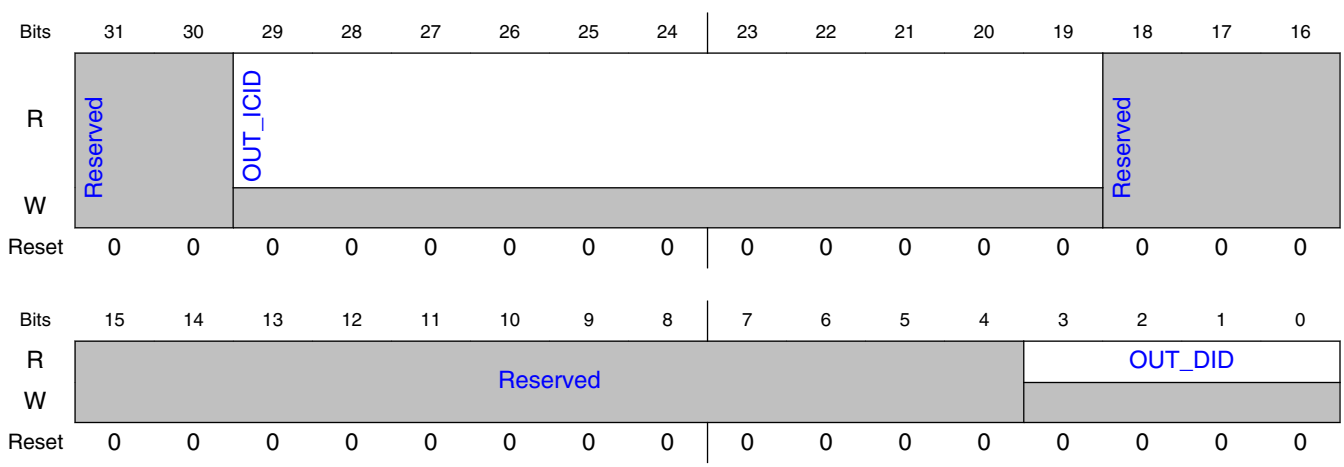
10.13.198 DECO0 Output DID Status Register (D0ODIDSR)

This register shows the DIDs and ICIDs available to the DECO for processing the current Descriptor. This register is written by the job queue controller.

10.13.198.1 Offset

Register	Offset	Description
D0ODIDSR	8824h	Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.198.2 Diagram



10.13.198.3 Fields

Field	Description
31-30	Reserved

Table continues on the next page...

Field	Description
—	
29-19 OUT_ICID	DECO Output ICID. When JRaDID[USE_OUT] =1, this is the ICID that is asserted during a data output DMA transaction. Note that the JRaDID[PRIM_ICID] value is asserted for job termination status writes and job descriptor update writes.
18-4 —	Reserved
3-0 OUT_DID	DECO Output DID. When JRaDID[USE_OUT] =1, this is the DID that is asserted during a data output DMA transaction. Note that the JRaDID[PRIM_DID] value is asserted for job termination status writes and job descriptor update writes.

10.13.199 DECO0 Math Register m_MS (D0MTH0_MS - D0MTH3_MS)

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 4 64-bit registers. Data is moved into these registers via LOAD, MATH and MOVE commands.

10.13.199.1 Offset

Register	Offset
D0MTH0_MS	8840h
D0MTH1_MS	8848h
D0MTH2_MS	8850h
D0MTH3_MS	8858h

10.13.199.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MATH_MS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MATH_MS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.199.3 Fields

Field	Description
31-0 MATH_MS	MATH register, most-significant 32 bits.

10.13.200 DECO0 Math Register m_LS (D0MTH0_LS - D0MTH3_LS)

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 4 64-bit registers. Data is moved into these registers via LOAD, MATH and MOVE commands.

10.13.200.1 Offset

Register	Offset
D0MTH0_LS	8844h
D0MTH1_LS	884Ch
D0MTH2_LS	8854h
D0MTH3_LS	885Ch

10.13.200.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MATH_LS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MATH_LS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.200.3 Fields

Field	Description
31-0 MATH_LS	MATH register, least-significant 32 bits.

10.13.201 DECO0 Gather Table Register 0 Word 0 (D0GTR0_0)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.201.1 Offset

Register	Offset
D0GTR0_0	8880h

10.13.201.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.201.3 Fields

Field	Description
31-0 —	Reserved

10.13.202 DECO0 Gather Table Register 0 Word 1 (D0GTR0_1)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.202.1 Offset

Register	Offset
D0GTR0_1	8884h

10.13.202.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDRESS_POINTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDRESS_POINTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.202.3 Fields

Field	Description
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).

10.13.203 DECO0 Gather Table Register 0 Word 2 (D0GTR0_2)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.203.1 Offset

Register	Offset
D0GTR0_2	8888h

10.13.203.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	E	F	Length													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Length															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.203.3 Fields

Field	Description
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 unused (i.e. Length and Address Pointer all 0s). 0 - Address Pointer points to a memory buffer. 1 - Address Pointer points to a Scatter/Gather Table Entry.
30 F	Final Bit. If set, this is the last entry of this Scatter/Gather Table. 0 - This is not the last entry of the SGT. 1 - This is the last entry of the SGT.
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.

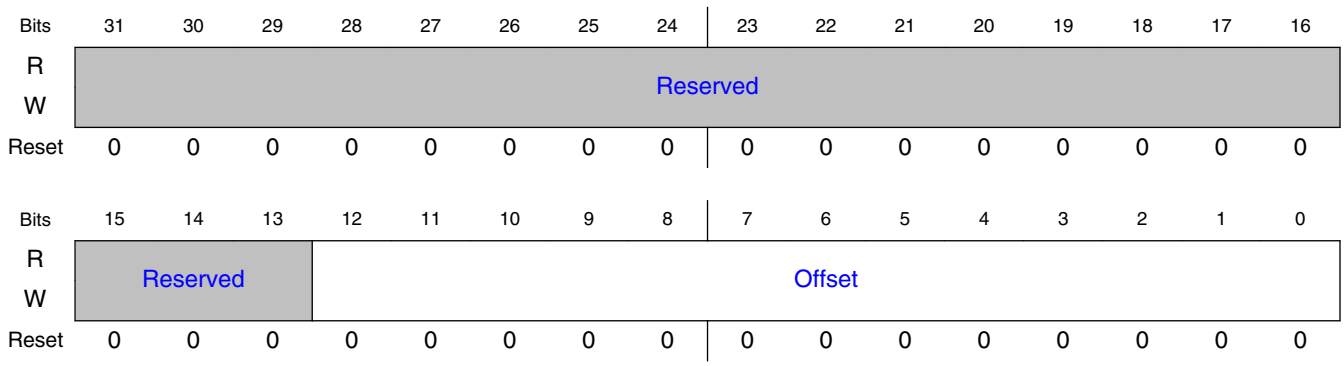
10.13.204 DECO0 Gather Table Register 0 Word 3 (D0GTR0_3)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.204.1 Offset

Register	Offset
D0GTR0_3	888Ch

10.13.204.2 Diagram



10.13.204.3 Fields

Field	Description
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.

10.13.205 DECO0 Scatter Table Register 0 Word 0 (D0STR0_0)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.205.1 Offset

Register	Offset
D0STR0_0	8900h

10.13.205.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.205.3 Fields

Field	Description
31-0 —	Reserved

10.13.206 DECO0 Scatter Table Register 0 Word 1 (D0STR0_1)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.206.1 Offset

Register	Offset
D0STR0_1	8904h

10.13.206.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDRESS_POINTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDRESS_POINTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.206.3 Fields

Field	Description
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).

10.13.207 DECO0 Scatter Table Register 0 Word 2 (D0STR0_2)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.207.1 Offset

Register	Offset
D0STR0_2	8908h

10.13.207.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	E	F	Length													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Length															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.207.3 Fields

Field	Description
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 unused (i.e. Length and Address Pointer all 0s). 0 - Address Pointer points to a memory buffer. 1 - Address Pointer points to a Scatter/Gather Table Entry.
30 F	Final Bit. If set, this is the last entry of this Scatter/Gather Table. 0 - This is not the last entry of the SGT. 1 - This is the last entry of the SGT.
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.

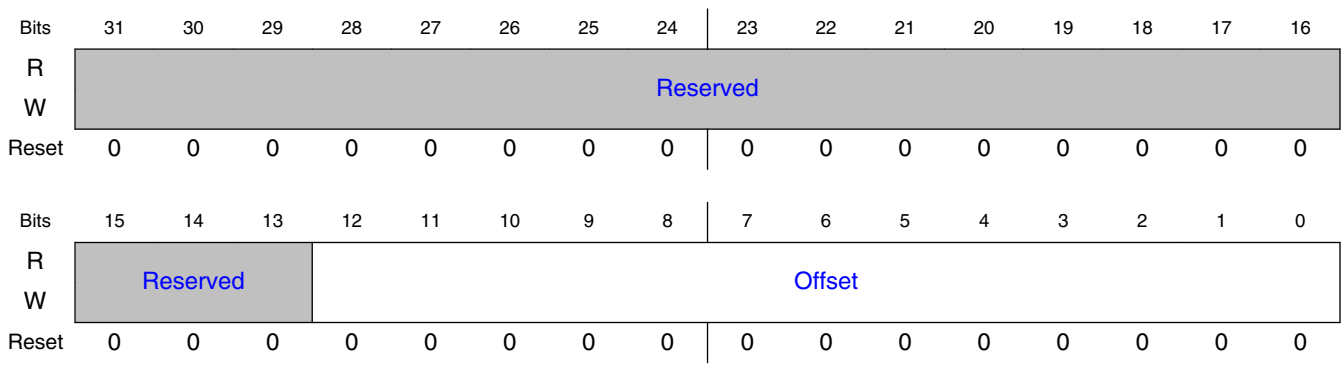
10.13.208 DECO0 Scatter Table Register 0 Word 3 (D0STR0_3)

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) 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.

10.13.208.1 Offset

Register	Offset
D0STR0_3	890Ch

10.13.208.2 Diagram



10.13.208.3 Fields

Field	Description
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.

10.13.209 DECO0 Descriptor Buffer Word a (D0DESB0 - D0DESB63)

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 above. For performance reasons, DECO doesn't execute the commands directly from the Descriptor Buffer. Instead, DECO executes commands from a four-word pipeline. 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. (They won't be executed if the job terminates or the pipeline is flushed as described below.) As a result, operations that modify the Descriptor Buffer may not have an immediate effect on the next few commands that execute. 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.

Note that the Descriptor Buffer is cleared between unrelated descriptors; that is, if two successive descriptors to execute in the same DECO do not share the same shared descriptor.

10.13.209.1 Offset

For a = 0 to 63:

Register	Offset
D0DESBa	8A00h + (a × 4h)

10.13.209.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DESBW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DESBW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.209.3 Fields

Field	Description
31-0 DESBW	Descriptor Buffer Word. In this instantiation of CAAM the descriptor buffer word registers hold up to 64 32-bit words of the descriptor currently processed by DECO. See Descriptors and descriptor commands for a definition of descriptor command and associated data words.

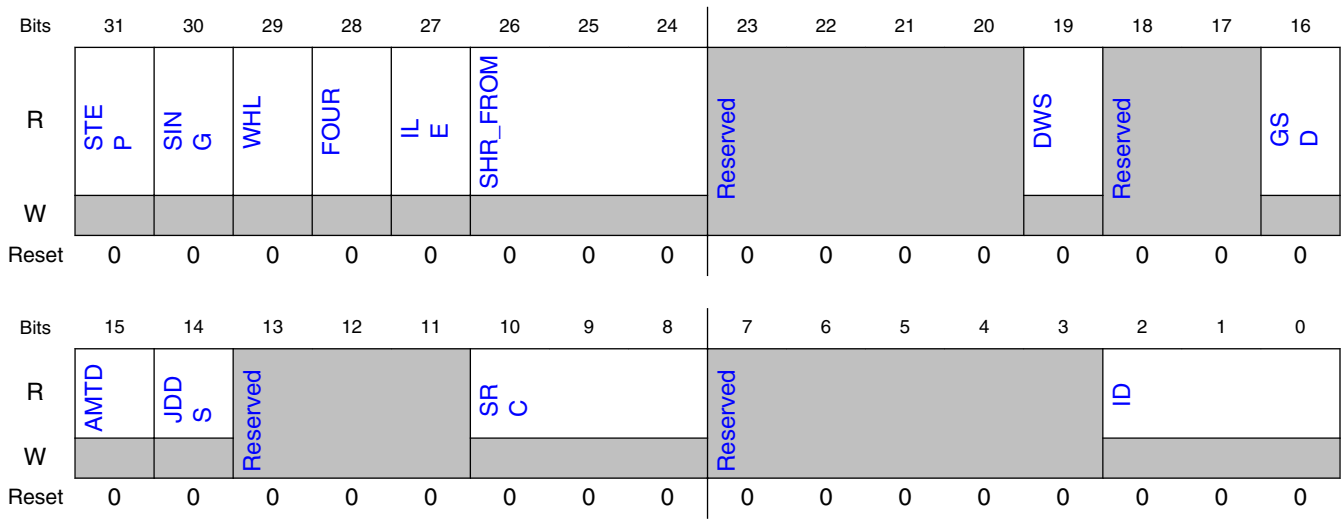
10.13.210 DECO0 Debug Job Register (D0DJR)

The DECO Debug 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 (see Register-based service interface). Note that the DECO Debug Job Register has the same format as the most-significant half of the DECO Job Queue Control Register. Note that this register is read-only.

10.13.210.1 Offset

Register	Offset	Description
D0DJR	8E00h	For DECO0

10.13.210.2 Diagram



10.13.210.3 Fields

Field	Description
31 STEP	Step. When in Single Step Mode, DECO will execute the next command in the descriptor. Note that protocols are a single step. 0 - DECO has not been told to execute the next command in the descriptor. 1 - DECO has been told to execute the next command in the descriptor.
30 SING	Single Step Mode. DECO has been told to execute this descriptor, including jumps to non-local destinations, in single-step mode. 0 - DECO has not been told to execute the descriptor in single-step mode. 1 - DECO has been told to execute the descriptor in single-step mode.
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the job queue controller (or by the processor that has control of DECO). This bit is set for certain Job Descriptors that are internally generated by CAAM. 0 - DECO has not been given the whole descriptor. 1 - DECO has been given the whole descriptor.
28 FOUR	Four Words. The job queue controller (or the processor that has control of DECO) has passed at least 4 words of the Descriptor to DECO. 0 - DECO has not been given at least four words of the descriptor. 1 - DECO has been given at least four words of the descriptor.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. Byte-swapping is controlled when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: • Input Data FIFO • Output Data FIFO • Auxiliary Data FIFO

Table continues on the next page...

Field	Description
	• Class 1 Context register • Class 2 Context register • Class 1 Key register • Class 2 Key register 0 - No byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer. 1 - Byte-swapping is performed for immediate data transferred to or from the Descriptor Buffer.
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).
23-20 —	Reserved
19 DWS	Double Word Swap. Double word swapping was set. 0 - Double Word Swap is NOT set. 1 - Double Word Swap is set.
18-17 —	Reserved
16 GSD	Got Shared Descriptor. A Shared Descriptor was already available in a DECO so the DECO didn't need to fetch the Shared Descriptor from memory. 0 - Shared Descriptor was NOT obtained from another DECO. 1 - Shared Descriptor was obtained from another DECO.
15 AMTD	Allow Make Trusted Descriptor. If this bit is a 1, then a Job Descriptor whose HEADER command has TDES=11b (candidate trusted descriptor) is allowed to execute. The AMTD 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 JRaDID Register. 0 - The Allowed Make Trusted Descriptor bit was NOT set. 1 - The Allowed Make Trusted Descriptor bit was set.
14 JDDS	Job Descriptor DID Select. Determines whether the SEQ DID or the Non-SEQ DID is asserted when reading the Job Descriptor from memory. 0 - Non-SEQ DID 1 - SEQ DID
13-11 —	Reserved
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS) and endian configuration bits) the DMA should use for bus transactions. When running descriptors via the IP bus (i.e. under the direct control of software), the job queue controller automatically sets this field to indicate a Job Ring source. 000 - Job Ring 0 001 - Job Ring 1 010 - Job Ring 2 011, 101, 110, 111 - Reserved 100 - RTIC
7-3 —	Reserved

Table continues on the next page...

Field	Description
2-0 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.

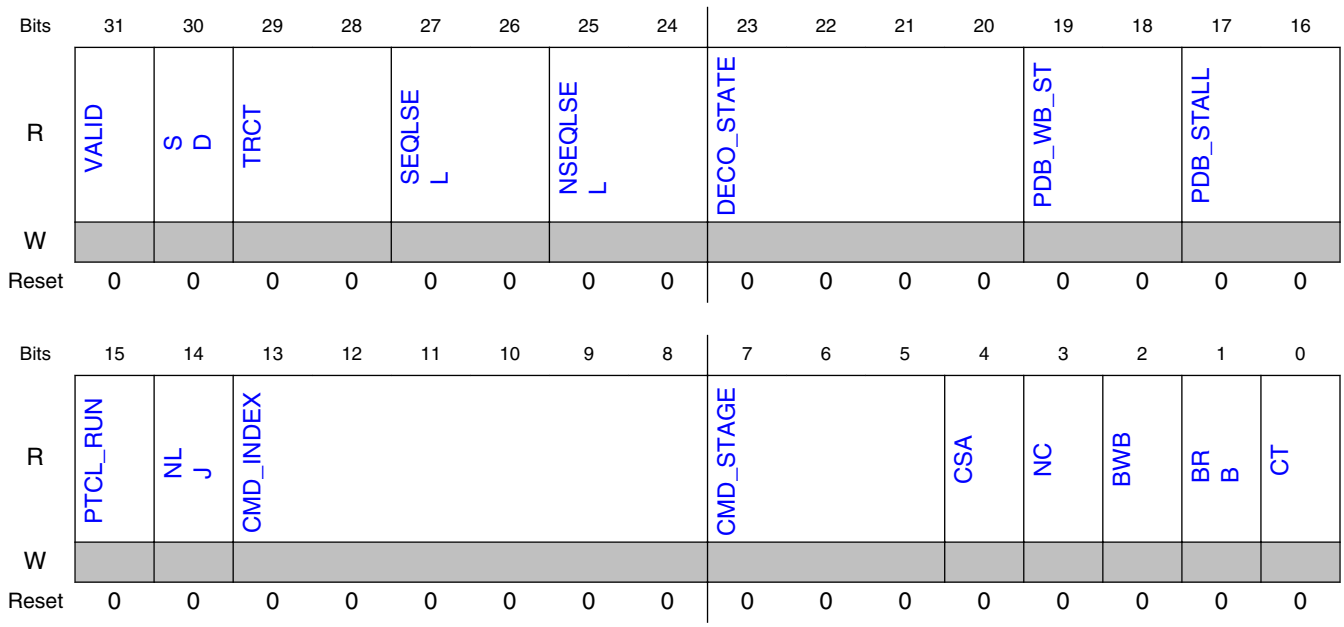
10.13.211 DECO0 Debug DECO Register (D0DDR)

The DECO Debug 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 (see Register-based service interface). Note that this register is read-only.

10.13.211.1 Offset

Register	Offset	Description
D0DDR	8E04h	For DECO0

10.13.211.2 Diagram



10.13.211.3 Fields

Field	Description
31 VALID	Valid. If VALID=1, there is currently a job descriptor running in this DECO. The descriptor has been loaded and has started executing and is still executing. 0 - No descriptor is currently running in this DECO. 1 - There is currently a descriptor running in this DECO.
30 SD	Shared Descriptor. The job descriptor that is running in this DECO has received a shared descriptor from another job descriptor. That is, some other job descriptor used this shared descriptor (in the same DECO or a different DECO), and this job descriptor is using the shared descriptor without having to load it from memory. In the case of SERIAL or WAIT sharing, then the keys were shared as well. If the SC bit was on and if the Shared Descriptor was shared within the same DECO (self-sharing), then the context was also shared. 0 - This DECO has not received a shared descriptor from another DECO. 1 - This DECO has received a shared descriptor from another DECO.
29-28 TRCT	DMA Transaction Count. This indicates how many outstanding external DMA transactions are pending. This is the total of reads and writes. In this instance of CAAM DECO is limited to, at most, 4 transactions (any combination of up to 2 reads and up to 2 writes).
27-26 SEQLSEL	SEQ DID Select. This indicates which type of DID is being used for SEQ commands: 01 - SEQ DID 10 - Non-SEQ DID 11 - Trusted DID
25-24	Non-SEQ DID Select. This indicates which type of DID is being used for Non-SEQ commands:

Table continues on the next page...

Field	Description
NSEQLSEL	01 - SEQ DID 10 - Non-SEQ DID 11 - Trusted DID
23-20 DECO_STATE	DECO State. The current state of DECO's main state machine.
19-18 PDB_WB_ST	PDB Writeback State. Lower two bits of the state machine that tracks the state of PDB writebacks.
17-16 PDB_STALL	PDB Stall State. The state of the machine that tracks the stalling of PDB writebacks. Used in conjunction with PDB_WB_ST. Used only if there is more than one DECO.
15 PTCL_RUN	Protocol running. PTCL_RUN=1 indicates that a protocol is running in this DECO. 0 - No protocol is running in this DECO. 1 - A protocol is running in this DECO.
14 NLJ	Took Non-local JUMP. If NLJ=1 the original job descriptor running in this DECO has caused another job descriptor to be executed. This is true for JUMP NON-LOCAL, SEQ IN PTR INLINE, and SEQ IN PTR RJD. 0 - The original job descriptor running in this DECO has not caused another job descriptor to be executed. 1 - The original job descriptor running in this DECO has caused another job descriptor to be executed.
13-8 CMD_INDEX	Command Index. If this DECO is currently executing a command, CMD_INDEX points to that command within the descriptor buffer.
7-5 CMD_STAGE	Command Stage. Each command executes in a number of steps, or stages. There are 8 possible stages. CMD_STAGE indicates which stage DECO has reached in the process of executing a command.
4 CSA	Command Stage Aux. A refinement of the CMD_STAGE stages. Some stages may be split into two substages, and CSA will indicate which of those two substages DECO has reached.
3 NC	No Command. This DECO is not currently executing a command. This can be because the descriptor isn't executing or DECO is doing a JUMP of some sort. 0 - This DECO is currently executing a command. 1 - This DECO is not currently executing a command.
2 BWB	Burster Write Busy. The WRITE machine in the Burster is busy. This means that the WRITE machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. STORE, SEQ STORE, FIFO STORE, and SEQ FIFO STORE commands use the WRITE machine. (The KEY command can also use the WRITE machine when obtaining a key modifier from the Secure Memory.) The WRITE machine is also used to update the Shared Descriptor HEADER when propagating DNRand by the Trusted State Machine to store a computed signature. 0 - The WRITE machine in the Burster is not busy. 1 - The WRITE machine in the Burster is busy.
1 BRB	Burster Read Busy. The READ machine in the Burster is busy. This means that the READ machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. LOAD, SEQ LOAD, FIFO LOAD, SEQ FIFO LOAD, and the KEY command all use the READ machine. The read to satisfy RIF in the Shared Descriptor HEADER also uses the READ machine. The SEQ FIFO STORE command can also use the READ machine when handling meta data. Jumping non-locally via any method will also use the READ machine. Commands that reference Scatter/Gather Tables will also cause the READ machine to be used to read the entries in the tables.

Table continues on the next page...

Field	Description
	0 - The READ machine in the Burster is not busy. 1 - The READ machine in the Burster is busy.
0 CT	Checking Trusted. This DECO is currently generating the signature of a Trusted Descriptor. This may be to sign, re-sign, or check the signature. 0 - This DECO is NOTcurrently generating the signature of a Trusted Descriptor. 1 - This DECO is currently generating the signature of a Trusted Descriptor.

10.13.212 DECO0 Debug Job Pointer (D0DJP)

The DECO Debug 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 (see Register-based service interface). Note that this register is read-only.

10.13.212.1 Offset

Register	Offset	Description
D0DJP	8E08h	For DECO0. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.212.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JDPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JDPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.212.3 Fields

Field	Description
63-32 —	Reserved
31-0 JDPTR	Job Descriptor Pointer.

10.13.213 DECO0 Debug Shared Pointer (D0SDP)

The DECO Debug 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 (see Register-based service interface). Note that this register is read-only.

10.13.213.1 Offset

Register	Offset	Description
D0SDP	8E10h	For DECO0. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR).

10.13.213.2 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SDPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SDPTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.213.3 Fields

Field	Description
63-32	Reserved

Table continues on the next page...

Field	Description
—	
31-0 SDPTR	Shared Descriptor Pointer.

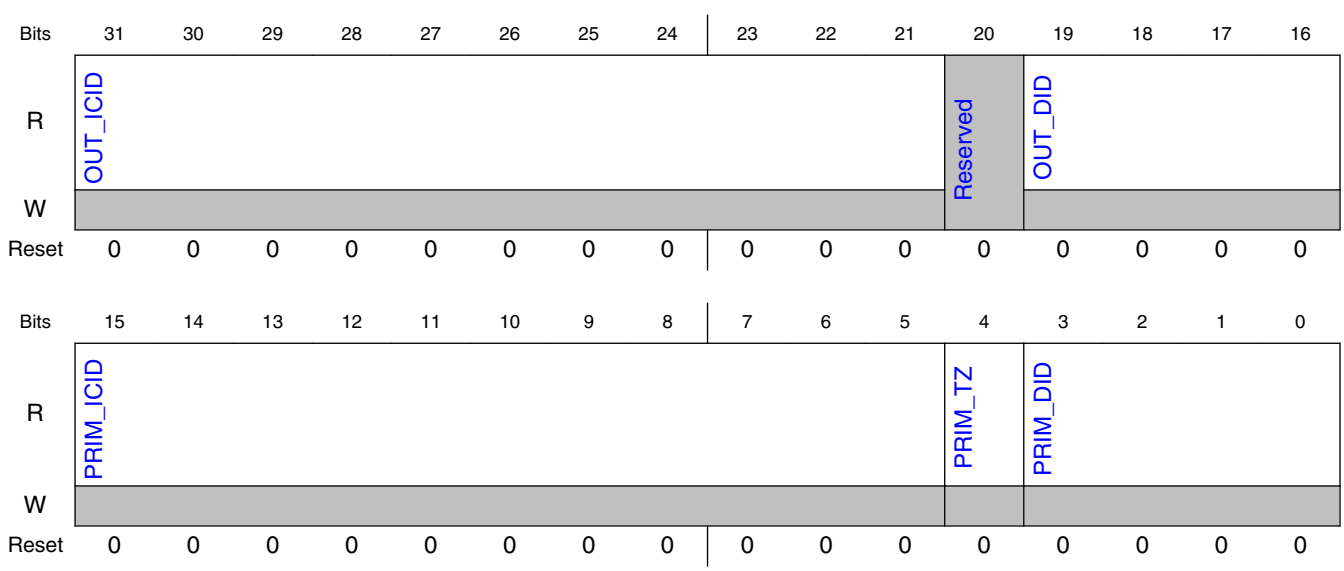
10.13.214 DECO0 Debug DID, most-significant half (D0DDR_MS)

The DECO Debug registers are intended to assist in debugging when a DECO appears to be hung. This register is read-only. 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 (see Register-based service interface).

10.13.214.1 Offset

Register	Offset	Description
D0DDR_MS	8E18h	For DECO0

10.13.214.2 Diagram



10.13.214.3 Fields

Field	Description
31-21 OUT_ICID	Output ICID. When JRaDID[USE_OUT]=0, the value in this field is unused. When JRaDID[USE_OUT]=1, this ICID value is asserted for external memory data writes, but not for reads or for job completion status writes or descriptor write-backs.
20 —	Reserved
19-16 OUT_DID	Output DID. When JRaDID[USE_OUT]=0, the value in this field is unused. When JRaDID[USE_OUT]=1, this DID value is asserted for all external memory data writes, but not for reads or for job completion status writes or descriptor write-backs.
15-5 PRIM_ICID	Primary ICID. When JRaDID[USE_OUT]=0, the value in this field indicates the ICID asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this ICID value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs.
4 PRIM_TZ	Primary TZ. When JRaDID[USE_OUT]=0, the value in this field indicates the TrustZone World value (PRIM_TZ=1 means SecureWorld) asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this TZ value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs. 0 - TrustZone NonSecureWorld 1 - TrustZone SecureWorld
3-0 PRIM_DID	Primary DID. When JRaDID[USE_OUT]=0, the value in this field indicates the DID value asserted for all external memory accesses. When JRaDID[USE_OUT]=1, this DID value is asserted for all external memory reads, and writes for job completion status and descriptor write-backs.

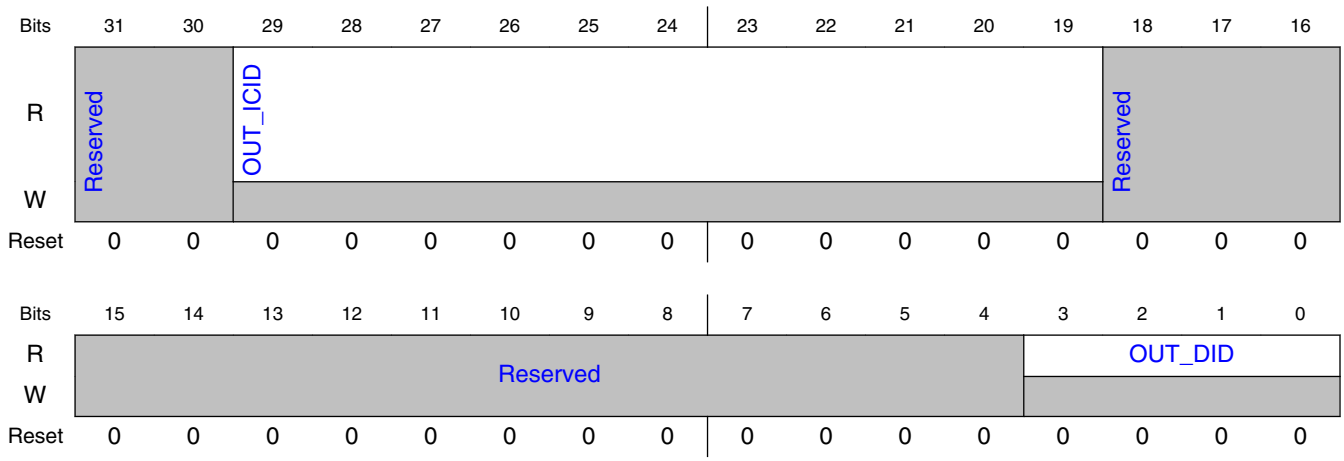
10.13.215 DECO0 Debug DID, least-significant half (D0DDR_LS)

The DECO Debug registers are intended to assist in debugging when a DECO appears to be hung. Note that this register is read-only. 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 (see Register-based service interface).

10.13.215.1 Offset

Register	Offset	Description
D0DDR_LS	8E1Ch	For DECO0

10.13.215.2 Diagram



10.13.215.3 Fields

Field	Description
31-30 —	Reserved
29-19 OUT_ICID	Output ICID. This is the ICID value that is asserted during DMA transactions related to data output operations.
18-4 —	Reserved
3-0 OUT_DID	DECO Output Domain Identifier. This DID value is asserted during DMA transactions related to data output operations.

10.13.216 Sequence Output Length Register (SOL0)

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). See SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the SOL register in Output Sequences. The SEQ OUT PTR command can be used to load the SOL register. The SOL Register can be read or written via the MATH Command (see SRC0

and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface) this register is accessible at the addresses shown above.

10.13.216.1 Offset

Register	Offset	Description
SOL0	8E20h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.216.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.216.3 Fields

Field	Description
31-0 SOL	Output Sequence Length. This value is used in output data sequences. SOL can also be used as a general purpose math register.

10.13.217 Variable Sequence Output Length Register (VSOL0)

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). See SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the

use of the VSOL register in Output Sequences. The VSOL Register can be read or written via the MATH Command (see SRC0, SRC1 and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface this register is accessible at the addresses shown above.Note that VSOL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSOL register, located at offset E34h.

10.13.217.1 Offset

Register	Offset	Description
VSOL0	8E24h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.217.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VSOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VSOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.217.3 Fields

Field	Description
31-0 VSOL	This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See UVSOL register.

10.13.218 Sequence Input Length Register (SIL0)

The Sequence In Length Register is used to specify the amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). See Section SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the SIL register in Input Sequences. The SIL Register can be read or written via the MATH Command (see SRC0 and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface this register is accessible at the addresses shown above. This register can also be loaded by the SEQ IN PTR command.

10.13.218.1 Offset

Register	Offset	Description
SIL0	8E28h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.218.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SIL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SIL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.218.3 Fields

Field	Description
31-0 SIL	This value is used in input data sequences. SIL can also be used as a general purpose math register.

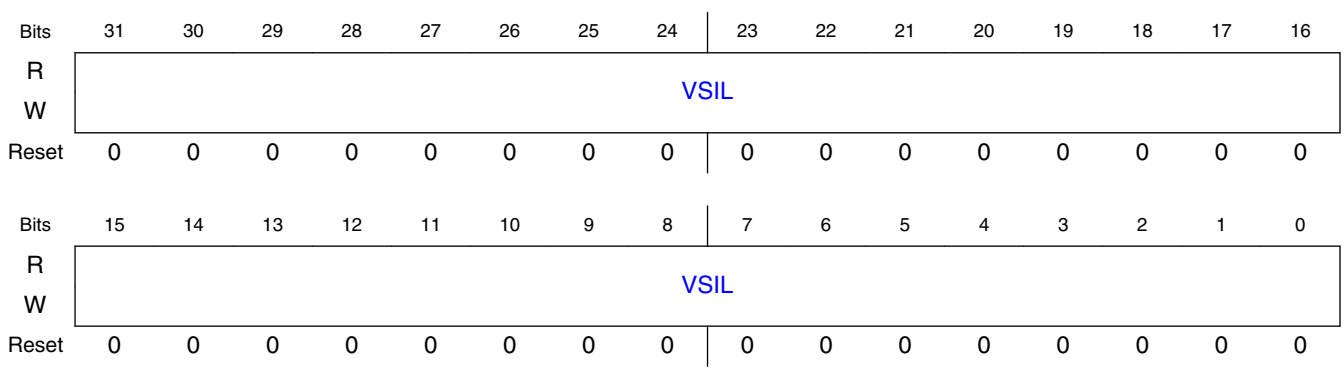
10.13.219 Variable Sequence Input Length Register (VSIL0)

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). See Section SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the VSIL register in Input Sequences. This register is also loaded when RIF is executed for a Shared Descriptor. The VSIL Register can be read or written via the MATH Command (see SRC0, SRC1, and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface) this register is accessible at the addresses shown above. Note that VSIL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSIL register, located at offset E38h.

10.13.219.1 Offset

Register	Offset	Description
VSIL0	8E2Ch	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.219.2 Diagram



10.13.219.3 Fields

Field	Description
31-0 VSIL	This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See UVSIL register.

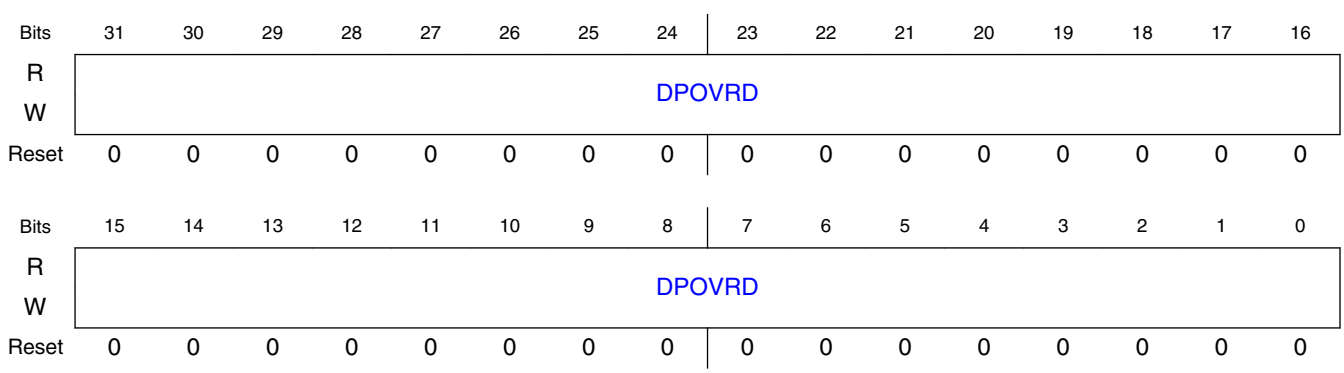
10.13.220 Protocol Override Register (D0POVRD)

This register can be read or written via the MATH Command (see SRC0, SRC1 and DST fields in MATH and MATHI Commands and it can be written via a LOAD IMMEDIATE command (see DST value 07h, Class=11 in LOAD commands). Note that DPOVRD can be used as a general purpose math register. When the DECO is under direct control of software (see Register-based service interface this register is accessible at the addresses shown above.

10.13.220.1 Offset

Register	Offset	Description
D0POVRD	8E30h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.220.2 Diagram



10.13.220.3 Fields

Field	Description
31-0 DPOVRD	DPOVRD can be used as a general purpose math register.

10.13.221 Variable Sequence Output Length Register; Upper 32 bits (UVSOL0)

VSOL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSOL register, located at offset E24h, and the most-significant 32 bits are accessible as the UVSOL register, located at offset E34h.

10.13.221.1 Offset

Register	Offset	Description
UVSOL0	8E34h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.221.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	UVSOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	UVSOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.221.3 Fields

Field	Description
31-0 UVSOL	This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See VSOL register. In some older versions of CAAM the UVSOL register did not exist, i.e. the VSOL register was only 32 bits. In those older versions when VSOL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVOL/VSOL and then 0 bits will be shifted in from the left.

10.13.222 Variable Sequence Input Length Register; Upper 32 bits (UVSIL0)

VSIL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSIL register, located at offset E2Ch, and the most-significant 32 bits are accessible as the UVSIL register, located at offset E38h.

10.13.222.1 Offset

Register	Offset	Description
UVSIL0	8E38h	For DECO0. Accessible only when RQD0 and DEN0 are asserted in DECORR.

10.13.222.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	UVSIL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	UVSIL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.13.222.3 Fields

Field	Description
31-0 UVSIL	This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See VSIL register. In some older versions of CAAM the UVSIL register did not exist, i.e. the VSIL register was only 32 bits. In those older versions when VSIL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVIL/VSIL and then 0 bits will be shifted in from the left.

Chapter 11
Secure Non-Volatile Storage (SNVS)

11.1 SNVS introduction

SNVS is a companion module to the CAAM module and serves as the SOC's central reporting point for security-relevant events such as the success or failure of boot software validation and the detection of security threat events. This security event information determines whether the SoC (hardware and software) is in the proper state to allow the CAAM to use persistent and ephemeral secrets. Based on configuration fuses and configured bits within registers, SNVS is able to detect a variety of security violation inputs and perform the configured policy enforcement actions.

SNVS incorporates both security and non-security functionality.

SNVS security functions:

- Monitor security events in the SoC, and respond as per configured security policy
- Protect certain security-critical data objects
  - Master Key - can be used by CAAM when encrypting or decrypting blobs
    - One-Time Programmable Master Key (OTPMK) from fuses can be used as the source of the Master Key
    - Zeroizable Master Key (ZMK) - can be used as the source of the Master Key
    - OTPMK XOR ZMK (Combined Master Key, CMK) - can be used as the source of the Master Key
  - Monotonic Counter (MC) - a monotonically increasing counter that can be used for replay detection
  - Secure Realtime Counter (SRTC) - realtime counter that can't be altered by untrusted software
- Preserve state of the data objects listed above (if the SNVS_LP power input is connected to an uninterrupted power supply)
  - ZMK value maintained when main SoC is powered off
  - MC value maintained when main SoC is powered off
  - SRTC continues to count when main SoC is powered off

SNVS non-security functions:

- Realtime Counter (RTC) - a software accessible realtime counter
  - RTC can be set to the value in the SRTC
- General Purpose Register - a set of registers used to hold 128 bits of data specified by software
  - If the SNVS_LP power input is connected to an uninterrupted power supply, the GPR value is maintained when main SoC is powered off
- Chip power-on/power-off - If the SNVS_LP power input is connected to an uninterrupted power supply and the Power On button input signal is connected to a power button external to the chip, logic within SNVS_LP can be used to wake the chip from a power down.

11.1.1 SNVS feature list

The following table summarizes the features of SNVS:

Table 11-1. SNVS feature list

Feature	Description	Links for Further Information
Security State Machine (SSM)	• Receives configuration inputs from SoC fuses • Receives security violation inputs from the various detectors in the chip • Tracks security state • Generates security state outputs to CAAM and other logic within the chip • Can request system hard reset in case of non-recoverable violation • Programmable high assurance counter (HAC) to control time delay before system hard reset request generation	Security state machine
Master key checking and control	• Performs validity checks for the one-time programmable master key before allowing CAAM to use it • Performs validity checks for the zeroizable master key before allowing CAAM to use it • Selects the device-specific master key value as the OTPMK, the ZMK, or the bit-wise exclusive OR of both the OTPMK and the ZMK	Configuring Master Key checking and control
Zeroizable master key (ZMK).	• The ZMK value can be programmed via software or via a hardware interface to CAAM's random number generator • The ZMK value can be selected for use in encapsulating or decapsulating CAAM blobs • The ZMK value is zeroized when a security violation occurs • If the SNVS_LP power input is connected to an uninterrupted power supply (see SNVS power domains), the value is maintained even when the main SoC is powered off.	Provisioning the Zeroizable Master Key
Secure real time counter (SRTC)	• The SRTC is driven by a dedicated clock that is functionally independent of the chip configuration. • The SRTC does not rollover. Instead the SRTC logic issues an alarm if the SRTC reaches its maximum value. • Programmable time alarm interrupt	SNVS_LP Secure Real Time Counter (SRTC)

Table continues on the next page...

Table 11-1. SNVS feature list (continued)

Feature	Description	Links for Further Information
	• Software can program the SRTC to generate an interrupt at a specific time. • If the main SoC is powered down at the programmed alarm time and the wake-up external alarm is enabled, the SRTC generates a wake-up alarm via an external pin. (Assuming that the SNVS_LP power input is connected to an uninterrupted power supply, see SNVS power domains) • The SRTC value is marked as invalid if an enabled SNVS security event is detected.	
Real time counter (RTC)	• The RTC is driven by a dedicated clock, which is off when the system power is down. • The RTC can be synchronized to the value of the Secure Real Time Counter. • Programmable time alarm interrupt	SNVS_HP Real Time Counter
Monotonic counter	• The monotonic counter can only increment. • The monotonic counter does not rollover. Instead the monotonic counter logic issues an alarm if the monotonic counter reaches its maximum value. • The monotonic counter value is marked as invalid if an enabled SNVS security event is detected. • If the SNVS_LP power input is connected to an uninterrupted power supply (see SNVS power domains), the monotonic counter value is retained even if the main chip is powered down.	Using the Monotonic Counter (MC)
General-purpose register	• The general-purpose register is available to software to store 128 bits of data. • The general-purpose register is zeroized when a security violation is detected. • If the SNVS_LP power input is connected to an uninterrupted power supply (see SNVS power domains), the general-purpose register value is retained even if the main chip is powered down.	Using the General-Purpose Register
Register access restrictions	• Some registers/values can be written only once per boot cycle.	privileged and non-privileged registers

Violation detection and reporting	• Detects (internal to the block) the following security violations:
	  • Scan exit
	  • Digital Low-Voltage Event for the LP domain
	  • Invalid OTPMK (ECC check failure)
	  • ZMK ECC check failure
	  • SRTC rollover
	  • MC rollover
	• Detects the following security violations:
	  • Software-reported violations
	  • 4 security violation inputs
	• Direct connections to CAAM to lock out access to the OTPMK/ZMK and force zeroization of sensitive information
	• Configurably triggers a device hard reset.
	• Configurably reports to software (interrupts) all security violation and functional events
		Security violation policy
		SNVS_LP security event policy

Wakeup from power off	• Input signal from off chip requests SNVS_LP to power on the main SoC (Assuming that the SNVS_LP power input is connected to an uninterrupted power supply (see SNVS power domains). • Hardware debounces the input signal using software-specified signal bounce characteristics	LP Wake-Up Interrupt Enable

11.1.2 SNVS functional description

The low-power section of SNVS is intended to be initialized at SNVS_LP POR, which normally occurs very rarely (when a new SNVS_LP power source is installed). At this time software initializes the SNVS as described in Reset and Initialization of SNVS. Once configured, much of the security functionality of SNVS operates automatically. However, there are a few SNVS security functions that involve interaction with software:

- reading the Secure Realtime Counter (SRTC) - this can be read, but not significantly altered[1], by runtime software
- reading or updating the Monotonic Counter (MC) - this can be read or incremented by runtime software
- SNVS also stores certain data whose integrity or confidentiality must be protected (e.g. secure real-time clock,monotonic counter,zeroizable master key).

One of the major security functions of SNVS is to sense security-related events on the device and control the device's security state per the software-configured security policy. Once SNVS is initialized, sensing these events and initiating a response is handled by hardware. Certain security-relevant events are detected within SNVS itself (e.g. power on reset, boot from SoC ROM, digital low-voltage event), whereas other security-related events are detected via sensors on or off-chip and passed to SNVS via input signals. For instance, these other sensors might detect voltage or temperature or clock frequency out of acceptable ranges.

SNVS implements several non-security features that involve software interaction:

- reading or writing the Realtime Counter (RTC) (This is a non-privileged operation.) - software can also instruct SNVS to load the current SRTC value into the RTC
- reading or writing the General Purpose Register (GPR) (Note that there may be a significant delay when reading or writing registers in the LP section if the LP clock is different from the HP clock.)

The following sections describe in more detail the operation of SNVS.

11.2 SNVS Structure

The SNVS incorporates several features that help to ensure the security of data stored in the device.

- A security state machine that responds to various security conditions
  - hardware security violation inputs (See SNVS_LP security event policy)
  - software-implemented security checks

1. The SRTC cannot be written after it has been locked by initialization software, but the clock rate can be adjusted until the adjustment bits are also locked.

- Security-state-machine derived outputs that tell the CAAM whether the device state is Trusted, Secure, NonSecure, or Fail
- Automatic zeroization of the Zeroizable Master Key if an enabled security event is detected
- Automatic zeroization of the General Purpose Register if an enabled security event is detected
- A digital low-voltage detector for the LP domain that alerts the security state machine when a low-voltage event causes a low-voltage detector bit to flip
- Validity flag that indicates if the value in the monotonic counter is valid
- Validity flag that indicates if the value in the secure realtime counter is valid

SNVS is organized as two major sub-modules:

- Low-Power Section of SNVS (SNVS_LP)

  The SNVS_LP section provides hardware that enables secure storage and protection of sensitive data. The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information.

  The SNVS_LP block implements the following functional units:
  - Zeroizable Master Key
  - Control and Status Registers
  - General Purpose Registers
  - Monotonic Counter
  - Secure Real Time Counter

  When the LP section is powered by an uninterrupted power supply, like a backup battery, the state of these registers is maintained even when the main chip power is off. (see SNVS power domains)

- High-Power Section of SNVS (SNVS_HP)

  The SNVS_HP section contains all SNVS status and configuration registers. It implements all features that enable system communication and provisioning of the SNVS_LP section. The SNVS_HP section also incorporates the security state machine, which controls the system security state, and the master key control block, which is responsible for checking and selecting the master key value. The SNVS_HP also incorporates the Zeroizable Master Key programming mechanism and the OTPMK logic.

  The SNVS_HP provides an interface between SNVS_LP and the rest of the system.

  The SNVS_HP block implements the following functional units:
  - IP Bus Interface
  - SNVS_LP Interface

- Security State Machine (SSM)
- Zeroizable Master Key Programming Mechanism
- Master Key Control block
- Real Time Counter with Alarm Control and Status Registers
- 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.

The following figure illustrates the structure of SNVS.

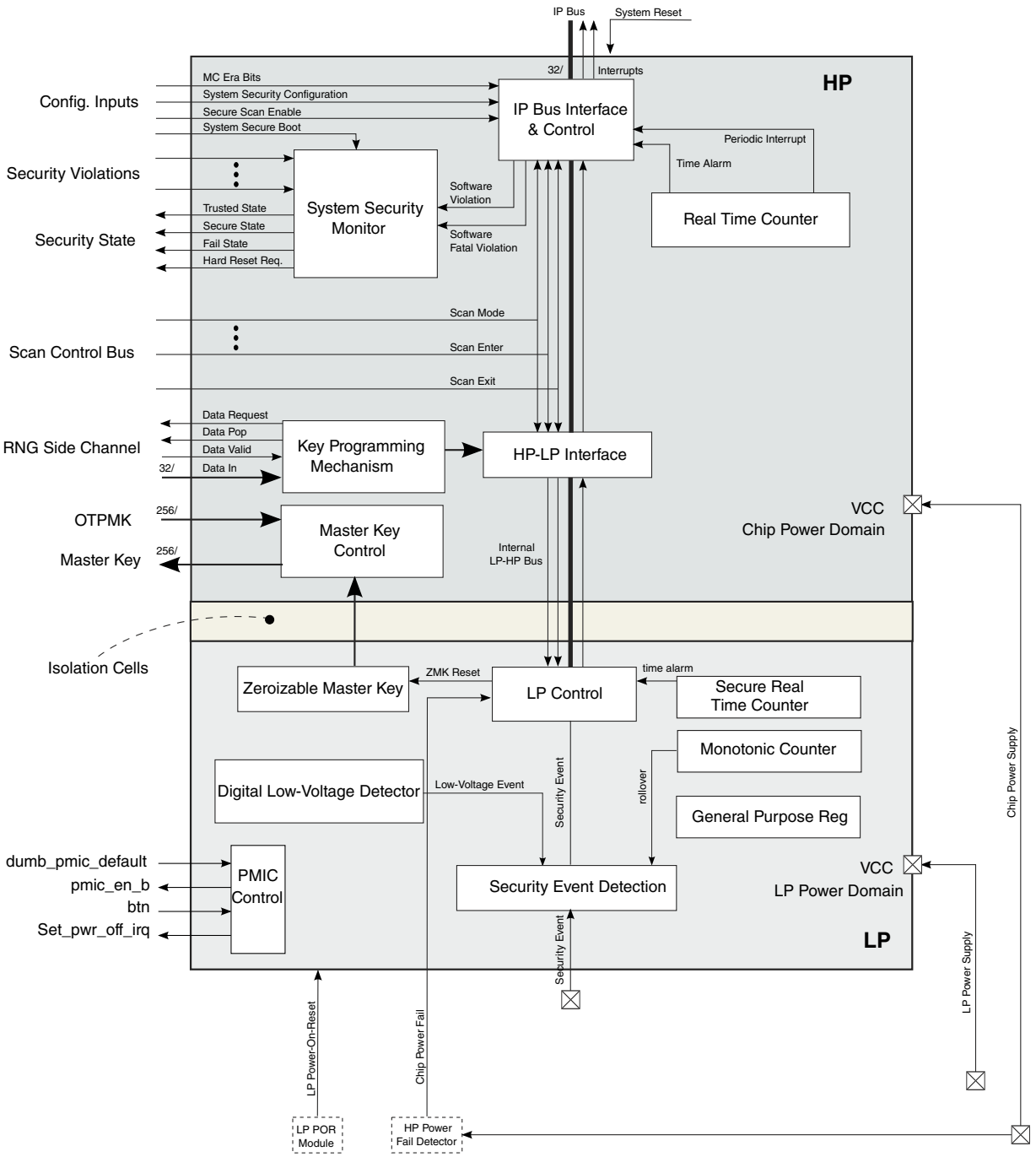


Figure 11-1. SNVS Block Diagram

11.2.1 SNVS power domains

In some versions of SNVS (including this version), the LP (Low Power) section is implemented in an independent power domain from the HP (High Power) section, and most other logic on the chip. Throughout the SNVS documentation whenever mention is

made of "always-on" logic, this assumes a version of SNVS that implements an independent power domain for the LP section, and that the power for this section is supplied by an uninterrupted power supply. The purpose for the independent power domain is so that data can be retained and certain logic can remain functional even when the main chip logic is powered down. But this is possible only if the LP domain remains powered via an uninterrupted power supply when the main chip power domain is powered off. Usually this uninterrupted power supply would be a battery, with possibly some power management logic to power the LP section from main power (and perhaps recharge the battery) when main power is on, and switch to battery power when the main power is off. In versions of SNVS with an independent LP power domain the LP section can be electrically isolated from the rest of the chip logic to ensure that its logic does not get corrupted when the main chip is powered down. If the battery runs down or is removed, an LP POR will occur when the LP section next powers up. Note that some OEMs may choose to connect LP power to HP/main chip power and dispense with a battery. In that case the SNVS will operate the same as an SNVS without an independent LP power domain. No state will be retained in the LP section when the chip is powered down, and an LP POR will occur whenever there is an HP POR.

11.2.2 Digital Low-Voltage Detector (LVD)

SNVS_LP incorporates a mechanism to detect interruptions of the SNVS_LP power supply that might cause the LP control, status, and secure counter values to change. The mechanism works as follows:

1. The LVD register (LPLVDR) is loaded with the known specific value 4173_6166h as part of the SNVS initialization process.
2. Subsequently, this register's value is continuously compared to the hardwired value 4173_6166h.
3. If the comparison indicates that any bit has changed, a low-voltage violation is asserted.

Digital low-voltage detection is always enabled and cannot be disabled. At LP POR this register is reset to all 0's, so the hardwired comparison fails and a low-voltage violation is reported. Therefore, before programming any feature in the SNVS the low-voltage violation should be cleared. The initialization software should write the proper value (4173_6166h) into LPLVDR (see SNVS_LP Digital Low-Voltage Detector Register (LPLVDR)) and should then clear the low-voltage event record in the LP status register (see SNVS_LP Status Register (LPSR)).

The following figure shows the LVD mechanism.

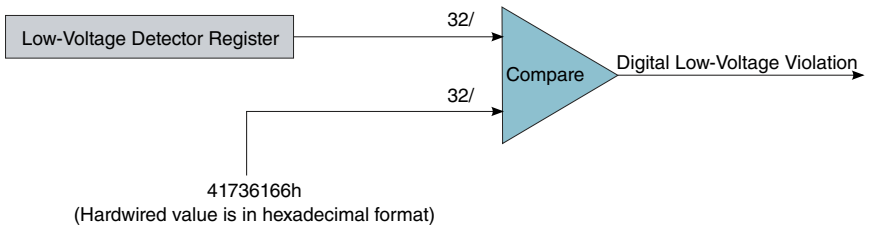


Figure 11-2. Digital low-voltage detector

11.2.3 SNVS clock sources

The SNVS has the following clock sources:

- System peripheral clock input. This clock is used by the SNVS's internal logic, for example, the Security State Machine. This clock can be gated outside of the module when the SNVS indicates that it is not in use.
- HP RTC clock. This clock is used by SNVS_HP real-time counter. This clock does not need to be synchronous with other clocks.

11.3 Security violation policy

SNVS is intended to detect, and potentially initiate a response to, various security-relevant events in the chip. The nature of the response to various types of security-relevant events can be set by configuring the SNVS security policy appropriately. SNVS detects certain security-relevant events either within its HP section, or via "Security Violation" inputs to the HP section. The following security-relevant events are detected within SNVS_HP hardware:

- OTPMK Hamming code error
- ZMK Hamming code error

The following security-relevant events are detected external to SNVS and are reported to SNVS_HP via "Security Violation" input signals:

- CAAM Security Violation
- JTAG Active
- Watchdog 2 reset
- Security Violation 3 (reserved)
- External Boot
- Security Violation 5 (reserved)

Software can also report a security violation to SNVS by writing into the SNVS_HP Command register (HPCOMR).

In addition to the SNVS_HP-detected security-relevant events mentioned, SNVS also detects certain other security-related events in the SNVS_LP section and reacts to them even when main SoC power is off:

- Digital Low-Voltage Event Violation
- SRTC Rollover Violation
- MC Rollover Violation

The SoC firmware specifies SNVS's response to security violations by configuring the SNVS_HP security policy. At boot time the SoC boot firmware also checks the status of the SNVS_LP section. If the LP section has not been securely initialized, or an enabled LP security event has occurred, or the SNVS_LP power source is interrupted, the status check will indicate that the state of the LP section is invalid, and the boot firmware will configure the SNVS_LP security policy. The configured security policy determines how security-related events affect the SoC security state, which is tracked by the SNVS Security State Machine as explained in the next section.

11.3.1 Security state machine

SNVS implements a security state machine (SSM) that tracks the security state of the SoC. The states and state transitions of the SSM affect certain security actions taken by SNVS and CAAM. For instance, in some security states CAAM will use a test key for blob operations, whereas in other security states CAAM will use a secret key. Some state transitions cause SNVS and CAAM to zeroize certain secret values.

The following figure and table describe the SSM's states and transitions.

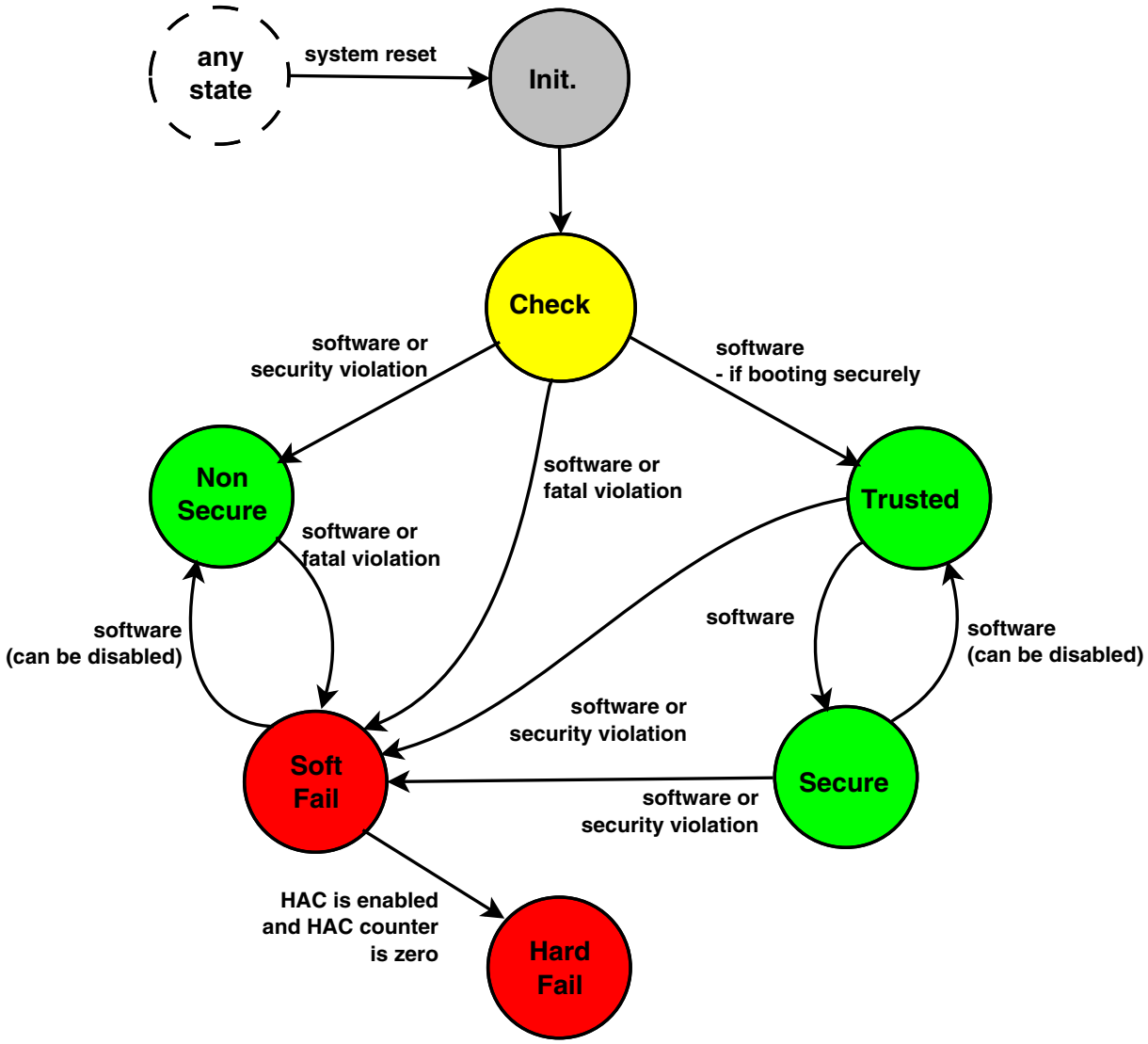


Figure 11-3. Security State Machine

Table 11-2. Security State Machine State Definitions

SSM State	What happens in this state	Security state signals sent to CAAM:	Transitions from this SSM State to:
Init *(System enters this state at main SoC 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.	Non-Secure (i.e. not Secure, not Trusted and not Fail)	• **Check state** - if the clocks are stable and fuse values can be read accurately the SSM transitions from Init state to Check state.
Check	System performs the check sequence, which consists of various hardware health checks. While in Check, the SNVS receives various inputs, including	Non-Secure (i.e. not Secure, not Trusted and not Fail)	• **Trusted state** - If software writes a 1 to the control bit in the HP Command Register while the SSM is in Check state, this initiates a transition to Trusted state. The transition is

Table continues on the next page...

Table 11-2. Security State Machine State Definitions (continued)

SSM State	What happens in this state	Security state signals sent to CAAM:	Transitions from this SSM State to:
	the fuse values from the SFP, security violation status, and a pass/fail indication from the boot firmware. SNVS registers cannot be programmed in this state, with the exception of several bits in the HP Command Register.		allowed if the system is booting from ROM internal to the chip, or if the System Security Configuration is Fab (000). Upon transition to Trusted state, SNVS_LP is reset if it was previously provisioned in the Non-secure state. • **Non-secure state** - If a non-fatal security violation is detected while in Check state the SSM transitions to the Non-secure state. • **Soft Fail state** - If a fatal violation is detected while in Check state the SSM transitions to the Soft Fail state.
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 state or Secure state.	Non-Secure (i.e. not Secure, not Trusted and not Fail)	• **Soft Fail state** - If a fatal violation is detected while in Non-Secure state the SSM transitions to the Soft Fail state.
Trusted	In this state, the trusted and secure mode indication signals are asserted. The fact that the SSM is in the Trusted state indicates that there are no hardware security violations and that HAB was satisfied with its security checks (including validating the signature over the next boot software). In this state SNVS signals to Cryptographic Acceleration and Assurance Module indicating that Cryptographic Acceleration and Assurance Module is allowed to use the Trusted/Secure state values of the blob master key, black key encryption keys, and trusted descriptor signing keys. Note that the PRI_BLOB bits in the Cryptographic Acceleration and Assurance Module Security Configuration register affect the derivation of the blob master key only when the SSM is in Trusted state.	Trusted, Secure	• **Secure state** - If software writes to the SSM_ST bit in the HP Command Register while SSM is in the Trusted state, this will trigger a transition to Secure state. • **Soft Fail state** - If the SSM detects a security violation condition while SSM is in the Trusted state, the SSM immediately (without clock) transitions to the Soft Fail state. If software writes to the HP Command register's SW_FSV or SW_SV bits while the SSM is in the Trusted state, the SSM will transition to Soft Fail state.
Secure	In this state, the secure mode indication signal is asserted and trusted mode indication signal is de-asserted. This indicates that Cryptographic Acceleration and Assurance Module is allowed to use the Trusted/Secure state values of the blob master key, black key encryption keys, and trusted descriptor signing keys. The only difference	Secure	• **Trusted state** - If software writes to the SSM_ST bit in the HP Command Register while SSM is in the Secure state, this will trigger a transition to Trusted state. (The Secure state to Trusted state transition can be

Table continues on the next page...

Table 11-2. Security State Machine State Definitions (continued)

SSM State	What happens in this state	Security state signals sent to CAAM:	Transitions from this SSM State to:
	between Secure and Trusted states is that the derivation of the blob master key is different. The PRI_BLOB bits in the Cryptographic Acceleration and Assurance Module Security Configuration register factor into the blob master key derivation only in Trusted state. This is intended to allow HAB and/or secure provisioning software to encrypt/decrypt blobs that cannot be encrypted or decrypted by runtime software.		disabled by setting the SSM_ST_DIS bit in the HP Command Register.) • **Soft Fail state** - If the SSM detects a security violation condition while SSM is in the Secure state, the SSM immediately (without clock) transitions to the Soft Fail state. If software writes to the HP Command register's SW_FSV or SW_SV bits while the SSM is in the Secure state, the SSM will transition to Soft Fail state.
Soft fail	Upon transitioning to Soft Fail the SNVS signals the Cryptographic Acceleration and Assurance Module to use the test master key for blobs and to zeroize the KEKs, TDSK and all key registers. The SNVS also generates an interrupt to inform software of the Soft Fail, and if configured to do so, starts the HAC_Counter countdown toward Hard Fail. The device can operate with the SSM in the Soft Fail state indefinitely; however, no operations involving the Cryptographic Acceleration and Assurance Module can occur unless the SNVS is transitioned from Soft Fail state to the Non-Secure state. Transition from Soft Fail to the Non-Secure state can be triggered by software via a write to the HP Command register SSM_ST bit, unless the HAC counter is actively counting or the transition to the Non-Secure state has been disabled via the SSM_SFNS_DIS bit. But the black keys and trusted descriptors that existed prior to the entry into Soft Fail will not be recoverable even in Non-Secure state and until the SoC is reset, the Cryptographic Acceleration and Assurance Module will be unable to perform blob operations with the secret master key. SNVS registers cannot be programmed in this state, except for several bits in the HP Command Register.	Fail	• **Hard Fail state** - If the HP Command Register HAC_EN bit is 1 when the SSM enters the Soft Fail state, the High Assurance Counter is loaded from the High Assurance Counter IV register and then the High Assurance Counter begins counting down. Software can stop this counter by writing to the HAC_STOP bit in the HP Command Register. If software fails to stop the High Assurance Counter before it counts down to zero the SSM transitions to the Hard Fail state. This acts as a "deadman switch" to reset the SoC if the software is hung. • **Non-Secure state**Software can initiate a transition from Soft-Fail state to Non-Secure state by writing a 1 to the SSM_ST bit in the HP Command Register, unless the transition has been disabled by setting the SSM_ST_DIS bit in the HP Command Register or the High Assurance Counter has been activated but not stopped, or a fatal security violation is active. In these cases the SSM stays in the soft fail state.
Hard Fail	Entering this state triggers the hard reset request output, which should be used in the system to perform a	Fail	• **Hard-Fail state** - One the SSM enters the Hard-Fail state it remains in the Hard-Fail state until the system is reset.

Table 11-2. Security State Machine State Definitions

SSM State	What happens in this state	Security state signals sent to CAAM:	Transitions from this SSM State to:
	hardware reset without the aid of software. SNVS registers cannot be programmed in this state.		

All HP security violation sources are classified into one of three categories: disabled, non-fatal security, and fatal security. Disabled security violations have no effect on the SSM. Fatal security violations always result in the SSM transition to the Soft Fail state. The non-fatal security violations result in a transition from Check to Non-Secure or from Trusted/Secure states to Soft Fail state.

- Disabled violation - SNVS records the violation but does not otherwise react to the violation.
- Non-fatal security violation - SNVS records the violation but does not cause security registers to be cleared
- Fatal security violation - SNVS records the violation and causes security registers to be cleared.

Regardless of the category all security violation events are recorded in the corresponding status registers.

11.3.2 SNVS interrupts, alarms, and security violations

When SNVS detects enabled security events, SNVS responds as dictated by the security policies pre-configured by software. This security policy configuration specifies under what circumstances SNVS asserts the following interrupt and alarm signals:

Table 11-3. Interrupts and alarms summary

Interrupt/violation	Security Event	Default configuration[1]	Configuration options
SNVS security interrupt	Security violation input asserted	Disable	Enable/disable
	SSM transitions to the soft fail state	Enable	-
	LP security violation asserted	Disable	Enable/disable
SNVS security violation	SSM transitions to the soft fail state	Enable	-
SNVS hard failure reset	SSM transitions to the hard fail state	Enable	-

1. Default behavior refers to the setting after Reset

11.3.3 Configuring SNVS's response to a security event

SNVS's response to hardware or software reporting a security violation depends on the state of the Security state machine, and the security policy that software has configured in SNVS. Software configures the SNVS security policy by writing to the following registers:

- SNVS_HP Security Interrupt Control Register (HPSICR) - enables or disables generating the security interrupt when specific security violation events are detected
- SNVS_HP Security Violation Control Register (HPSVCR) - configures specific security violation event inputs as fatal or non-fatal or disabled
- SNVS_HP High Assurance Counter IV Register (HPHACIVR) - sets the delay between the occurrence of an SSM transition from soft fail state to hard fail state
- SNVS_HP Lock Register (HPLR) - locks part or all of the security configuration set in SNVS_HP
- SNVS_LP Control Register (LPCR) - configures certain LP security event responses:
  - whether GPRs are zeroized in response to a security event
  - whether an LP low-voltage event alerts the PMIC
  - whether a wake-up interrupt is requested when an LP section security event occurs
  - whether the SRTC stops counting when a security violation occurs
- SNVS_HP Security Violation Control Register (HPSVCR) - configures whether specific HP section security violation event inputs cause an LP section security violation, which zeroizes sensitive data in the LP section
- SNVS_LP Lock Register (LPLR) - locks part or all of the security configuration set in SNVS_LP

If the SNVS SSM is in the Trusted or Secure state, the occurrence of a security violation triggers a transition to the Soft Fail state. The transition to Soft Fail state is reported to CAAM, which will also take certain response actions.

The consequences of an SSM transition to Soft Fail include:

- Zeroization of the JDKEK, TDKEK and TDSK values in CAAM
  - Zeroization of the JDKEK effectively zeroizes all normal Black Keys
  - Zeroization of the TDKEK effectively zeroizes all Trusted Black Keys
  - Zeroization of the TDSK invalidates all Trusted Descriptors
- Zeroization of the ZMK value in SNVS_LP
  - Zeroization of the ZMK effectively zeroizes all blobs encrypted using the ZMK or using ZMK XOR OTPMK
- Zeroization of the GPR value in SNVS_LP

    - Zeroization of the GPR protects the confidentiality of key or other secret data stored in the GP register
- Lock out of the master key (OTPMK, ZMK or OTPMK XOR ZMK) until the next POR (and successful secure boot)
    - Lock out prevents Trusted or Secure blobs from being decrypted until POR (and successful secure boot)

Note that software can continue to run following a transition to Soft Fail, and both SNVS and CAAM remain operable. However, it is inadvisable to continue normal operation until the SoC has been reset. Whatever event triggered the transition from Secure or Trusted to Soft Fail state may have left the SoC in a vulnerable condition. Also, as explained below, after such a transition certain CAAM security functions will be inoperable or unsecure.

- Attempting to decrypt Secure and Trusted blobs following a transition to Soft Fail will produce an error indication. (Since the Master Key is unavailable, the signature on the blobs will be incorrect.)
- Attempting to use pre-Fail-transition CCM-encrypted Black Keys will produce an error indication. (The JDKEK and TDKEK values will be incorrect now that they have been zeroized, consequently the CCM cryptographic signature will be incorrect).
- Attempting to use pre-Fail-transition Trusted Descriptors will produce an error indication. (The TDSK value will be incorrect now that it has been zeroized, consequently the cryptographic signature will be incorrect).
- Using pre-Fail-transition ECB-encrypted Black Keys will not produce an error indication because there is no cryptographic signature on ECB-encrypted Block Keys. (Using these keys will simply yield an incorrect result.)
- Encrypting blobs will not produce an error, but these blobs will not be effectively protected. (Since the Master Key is unavailable, the test key will be used.)
- Generating either CCM-encrypted or ECB-encrypted Black Keys will not produce an error, but these Black Keys will not be effectively protected. (Since the JDKEK and TDKEK have been zeroized, Black Keys will be encrypted with a known value.)

Consequently, operation following a transition from Secure or Trusted to Soft Fail state should be limited to recovery and debugging. In response to a Soft Fail, a security violation interrupt service routine should read the SNVS HP_Status Register and HP_Security Violation Status Register and write their values to NVRAM to log the occurrence and root cause of the security violation.

11.3.4 SNVS_LP security event policy

The tables below provide information about the SNVS_LP security event policy.

Table 11-4. SNVS_LP security events

Security event	Default behavior[1]	Configuration options[2]	Comments
Internal to LP Sources[3]			
SRTC Rollover Violation	Disable	• Enable • Disable	Asserted when SRTC is enabled and it reaches maximal value (all ones)
MC Rollover Violation	Disable	• Enable • Disable	Asserted when MC is enabled and it reaches maximal value (all ones)
From HP Section			
LP software violation	Enable	• Enable • Disable	Asserted by software
HP Security Violation Inputs SV0, SV1, SV2, SV4, (See SNVS_LP security event policy)	Disable ../inst	• Enable • Disable	Asserted on the Security Violation Port

1. Default behavior refers to the setting after LP POR, before the LP Security Violation policy is configured.
2. The configuration is set in the LPSECR and LPSVCR registers. Once set, the configuration can be locked to prevent further changes.
3. These security events can also be asserted and detected in the system power-down mode.

When an LP security violation is generated, the violation is reported to the SNVS_HP section. The source of the violation is recorded in the LP status register.

11.3.5 High Assurance Counter

If a security event occurs that transitions SNVS to the Soft Fail state, the software that is intended to respond to this transition may fail to act because it has become corrupted. SNVS implements a software "deadman" switch that is intended to deal with this case. The deadman switch consists of the High Assurance Counter (HAC) and the various other registers and register bit fields that control the HAC.

If enabled, the HAC begins counting down toward 0 when the SNVS Security State Machine (SSM) transitions to Soft Fail state. If software does not stop the count before it reaches 0, the security state machine will transition to the Hard Fail state. Transitioning to Hard Fail state is intended to cause the chip to reset.

If the HAC will be used, sofware should write the HAC initial value to the High Assurance Counter IV Register and then write a 1 to the HAC_LOAD (High Assurance Counter Load) bit and a 1 to the HAC_EN (High Assurance Counter Enable) in the HP Command Register. This will load the initial value into the HAC Register and enable the HAC for counting. Once the HAC register is initialized and enabled, software should write a 1 to the HAC_L (High Assurance Configuration Lock) bit in the HP Lock Register to prevent malicious or corrupted software from altering the HAC configuration.

If a security event causes the SSM to transition to SOFT FAIL state the HAC will start counting, one count per system clock, from the initial value down to 0. While the SSM is in SOFT FAIL state software can halt the count by writing a 1 to the HAC_STOP bit in the HP Command Register. If software later writes a 0 to HAC_STOP the count will resume at the current value in the HAC. If software writes a 1 to the HAC_CLEAR bit in the HP Command Register, the HAC will be cleared, regardless of the state of the SSM. If the SSM is in the Soft Fail state at that time the SSM will immediately transition to the Hard Fail state. If the SSM is in any other state when the HAC is cleared the SSM will transition to Hard Fail immediately after transitioning to Soft Fail (assuming the HAC is enabled).

11.4 Runtime Procedures

SNVS implements a number of features that are intended to be accessed by software at runtime (as opposed to accessed at boot time). These features include:

- Real Time Clock (see SNVS_HP Real Time Counter)
- Secure Real Time Clock (see SNVS_LP Secure Real Time Counter (SRTC))
- General Purpose Register (see Using the General-Purpose Register)
- Monotonic Counter (see Using the Monotonic Counter (MC))

Procedures for using these features are described in the following sections.

11.4.1 Using SNVS Timer Facilities

SNVS incorporates timer facilities that can optionally generate an interrupt at a specified time. As described in the following sections, SNVS_HP incorporates a Real Time Counter that is available for general use, and SNVS_LP incorporates a Secure Real Time Counter intended for security applications.

11.4.1.1 SNVS_HP Real Time Counter

SNVS_HP implements a real time counter that can be read or written by any application; it has no privileged software access restrictions. When the chip is powered down the RTC is not active and it is reset at chip POR. The RTC can be used to generate a functional interrupt request either at a specific time, or at a specific frequency, or both. To generate an interrupt request at a specific time HPTA_EN is set to 0, the desired time is written to HPTA_MS and HPTA_LS and then HPTA_EN is set to 1. HPTA_EN, HPTA_MS and HPTA_LS can be written by any software that has access to SNVS registers; there are no

privileged access restrictions. The counter can be synchronized to the SNVS_LP SRTC by writing to the HP_TS bit of SNVS_HP Control Register. This is particularly useful if the SNVS_LP is powered from an uninterrupted power source because the RTC can then be set from a chip-internal time source.

11.4.1.2 SNVS_LP Secure Real Time Counter (SRTC)

SNVS_LP implements a secure real time counter. The SRTC differs from the Realtime Counter (RTC) implemented in the HP section in the following ways:

- If the SNVS_LP power input is connected to an uninterrupted power supply, the SRTC retains its state and continues counting even when the main chip is powered down.
- When the RTC reaches its maximum value it simply rolls over to all 0s. The SRTC is a non-rollover counter, which means that the SRTC does not rollover to all zeros when it reaches the maximum value of all ones. Instead, a time rollover indication is generated to the SNVS_LP security event monitor, which generates a security violation and interrupt.
- The RTC can be written at any time. The SRTC can be locked so that a new value cannot be written into the SRTC.
- The RTC can be synchronized to the SRTC by writing to the HP_TS bit of SNVS_HP Control Register.
- The SRTC can be calibrated so that its count frequency tracks some other time source.
- The SRTC can be marked as valid or invalid, so software knows whether the SRTC value can be trusted.
- The SRTC can be configured to capture the time of a security violation.

The SRTC is intended for security applications that require real time. Some examples are:

- data rights management schemes in which the rights expire at a certain date and time
- audit logs that record the date and time of security-relevant events
- security protocols that rely on real time to ensure freshness or for event ordering

To ensure that the SRTC cannot be modified in order to circumvent time-based security policies, the SRTC 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. If the SRTC will be used for security purposes, at SNVS_LP POR trusted software (e.g. boot firmware) should check if SRTC is enabled and valid (SRTC_ENV=1). If SRTC_ENV=0 the trusted software should set the SRTC from a trustworthy time source and then set SRTC_ENV, SRTC_SL and SRTC_HL to 1. When SRTC_SL=1 the SRTC cannot be written until main chip reset. When SRTC_HL=1 the SRTC cannot be written until SNVS_LP POR.

The SRTC can be used to generate a functional interrupt request at a particular time by setting the desired time in the LP Time Alarm register (LPTAR) and setting LPTA_EN to 1. Note that the functional interrupt request is shared by both the LP time alarm and the HP time alarm, so the interrupt handler for the SNVS functional alarm should check the HPTA bit in the HPSR and/or the LPTA in the LPSR to distinguish between these two alarm sources.

The SRTC counting speed can be adjusted to match the speed of some other time source via a calibration mechanism. When the LPCALB_EN, LPCALB_SL, and LPCALB_HL bits are 0 a value can be written into the LPCALB_VAL field of the LP Control Register. When LPCALB_EN is 1, every 32768 ticks of the SRTC the adjustment value specified in the LPCALB_VAL field will be added to the SRTC. If needed, at LP POR the LPCALB_VAL field should be set by trusted software and then LPCALB_EN set to 1. Whether the SRTC calibration mechanism is used or not, trusted software should set LPCALB_SL, and LPCALB_HL bits to 1 to prevent misuse of the calibration mechanism.

11.4.1.3 RTC/SRTC control bits setting

All SNVS registers are programmed from the register bus, consequently any software-initiated 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, the following values can be changed only when the corresponding function is disabled.

Table 11-5. RTC/SRTC synchronized values list

Function	Value/register	Control bit setting
HP section		
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
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

Table 11-5. RTC/SRTC synchronized values list

Function	Value/register	Control bit setting
		LPTA_EN = 1 : LPTAR **cannot** be programmed

Use the following steps 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 IP clock 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.

11.4.1.4 Reading RTC and SRTC values

Software should follow the following procedure to ensure that it has read correct data from the RTC (HPRTCMR and HPRTCLR) and SRTC (LPSRTCMR and LPSRTCLR) registers:

- Read the most-significant half and the least-significant half of the RTC/SRTC and then read both halves again. If the values read are the same both times, the value is correct.
- If the two consecutive pairs of reads yield different results, perform two more reads.

The worst case scenario may require three sessions of two consecutive pairs of reads. There are several reasons that the values may be incorrectly read initially:

- Synchronization issues between the RTC/SRTC clock and the system clock
- Since the counter continues to increment, there may be a carry from the least-significant 32-bits to the most-significant bits in between reading the two halves of the counter

11.4.2 Using Other SNVS Registers

The sections below describe how to use the General Purpose Register.Monotonic Counter.The sections below describe how to use the General Purpose Register and the Monotonic Counter.

11.4.2.1 Using the General-Purpose Register

SNVS implements a 128-bit general-purpose register allows software to store a small amount of data. To maintain backward compatibility with versions of SNVS that implement only a 32-bit general purpose register, the most-significant word of the general purpose register is aliased to the original legacy address, and to maintain backward compatibility with versions of snvs_module_name that implement a 128-bit general purpose register, the most-significant half of the general purpose register is aliased to the previous legacy address address. The data in the GPR will be retained during system power-down mode as long as the SNVS_LP remains powered by an uninterrupted power source.If LP Control Register GPR_Z_DIS bit is 0, the GPR will be zeroized if an enabled security event occurs.

11.4.2.2 Using the Monotonic Counter (MC)

The following figure shows the MC and its rollover security violation.

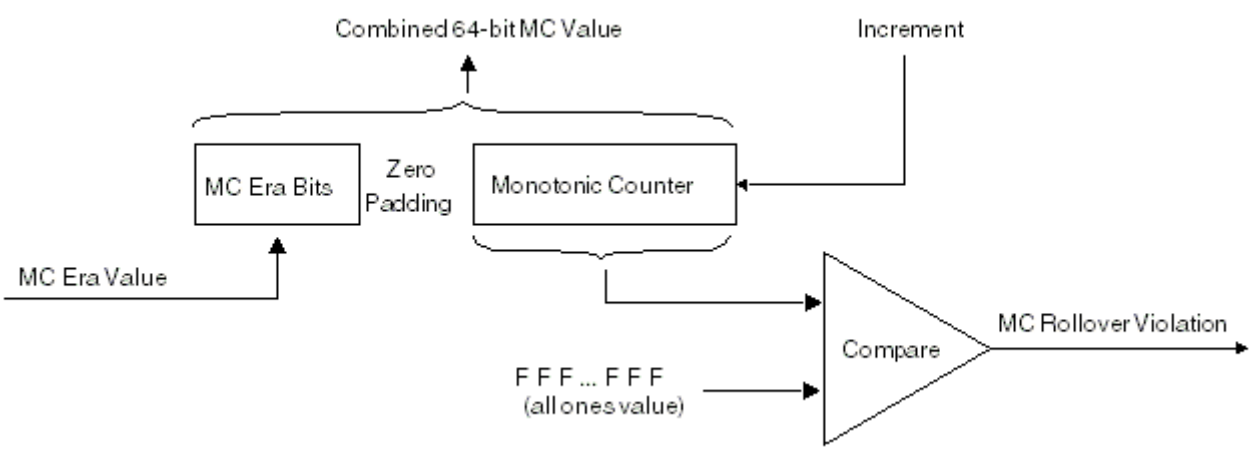


Figure 11-4. SNVS_LP monotonic counter

Some security applications require a monotonic counter (MC) that cannot be returned to any previous value during the product's lifetime. For instance, the MC can be used to detect replay of a cryptographic blob by including the current monotonic counter value in the blob when it is encrypted, and checking this value against the current value in the MC register when the blob is decrypted. If more than one blob must be replay-protected, when any blob is to be updated the MC value would be incremented and all the blobs would be refreshed with the updated MC value. Alternatively, software monotonic counters could be maintained in a "monotonic counter blob", which would be replay-protected via the hardware MC. In this case any time that a blob is to be updated its software monotonic counter would be incremented, the MC would be incremented, and the monotonic counter blob would be refreshed with the updated value of the software monotonic counter and the updated MC value. If the application can tolerate a limited window for replay, the overhead could be reduced by only updating the monotonic counter blob once an hour, or perhaps once a day. Software could create even more elaborate anti-replay mechanisms using a tree of monotonic counter blobs, with the hardware MC at the root.

Because the MC cannot be allowed to repeat a value, it cannot set to a lower value than it currently holds. In the unlikely case that it reaches its maximum value, it does not rollover. Instead, a monotonic counter rollover indication is generated to the SNVS_LP security event monitor. This generates a security violation to the SSM and an interrupt to the host processor.

The MON_COUNTER fields of the MC register are implemented in flip flops within the LP section. If LP power is lost, the MON_COUNTER value will be lost. To preserve monotonicity in this event, the most significant bits of MC (the MC_ERA_BITS field) are derived from fuses. This ensures that the MC_ERA_BITS value is preserved across LP section power failures. The next time that the chip powers up following an LP section power failure, the chip's boot firmware will note that the monotonic counter value is invalid and will blow another of the fuses that drive the MC_ERA_bits field. This will result in a larger value in the MC_ERA MC field and since this field forms the most-significant bits of MC, this guarantees that the new value of MC is greater than any of its past values.

11.5 Configuring Master Key checking and control

During cryptographic blob operations CAAM uses a key derivation function to create a 256-bit blob key encryption key (BKEK). The BKEK is used to encrypt or decrypt a 256-bit random key that is actually used to encrypt or decrypt the content of the blob. Note that the BKEK is nonvolatile so that blobs generated during one power-on cycle can be decrypted during later power-on cycles. One of the inputs to the BKEK key derivation

function is the Master Key. When CAAM is in the non-Secure or Fail mode, a test value is used for this Master Key. But when CAAM is in the Trusted or Secure mode the key value used is the Master Key supplied by the SNVS. Software configures SNVS via the LP Master Key Control Register's MASTER_KEY_SEL (Master Key Select) field so that the Master Key supplied to CAAM comes from one of the following sources:

- One Time Programmable Master Key (OTPMK): This value is permanently burned into fuses. The 256-bit value includes an embedded 9-bit Hamming code so that hardware can verify the integrity of the key. This code is capable of detecting all single, double, and triple bit errors in the key value. The construction of the Hamming code is described in Error Code for the OTPMK. The hardware checks for an all-zero value of the OTPMK. If the OTPMK Hamming code check fails or the OTPMK is not programmed at all (all 0s), the hardware will report a "bad key" error, which will prevent the chip from reaching the Trusted/Secure state.
- Zeroizable Master Key (ZMK): This value is stored in the ZMK registers of the low power (LP) section of the SNVS. The ZMK value is retained when the main SoC power is off because power is supplied to the SNVS LP section from an uninterrupted power source. The ZMK value is automatically erased in response to fatal security violations. OEMs have two options for provisioning a ZMK:
  - Software Programming: Software can write directly to the SNVS's ZMK registers. Once software has finished writing all of the ZMK registers, software sets the ZMK_Valid bit. The value to be written into the ZMK can be determined by any appropriate method. For instance, a random value could be obtained from the CAAM's RNG, or software could program the ZMK registers with a secret value shared among the OEM's devices. Unlike the OTPMK, the ZMK value does not include an embedded Hamming code. Instead, additional ECC protection bits are automatically generated by hardware and stored in the SNVS LP section.
  - Hardware Programming: SNVS and CAAM can be configured so that a 256-bit random value is generated in the CAAM's random number generator and loaded directly into the SNVS's ZMK registers via a private hardware interface. Additional ECC protection bits are automatically generated by the SNVS and stored in the SNVS LP section along with the ZMK.
- Combined Master Key (CMK): SNVS can be configured so that the Master Key is the bitwise XOR of the OTPMK registers and the ZMK registers. This bitwise XOR is called the CMK.

When SNVS detects a fatal security violation, SNVS signals the CAAM to switch to the Test key instead of the Master Key value when encrypting or decrypting blobs. Until the next successful secure boot CAAM will not be able to decrypt any blobs that were generated while CAAM was in Trusted or Secure mode. Note that the Master Key is used by the CAAM only as a key derivation key, and only during blob operations. It is never

used to directly encrypt or decrypt user selected data. Within the CAAM the Master Key value is combined with a number of other inputs to derive a blob key encryption key (BKEK). During blob creation the BKEK is used to encrypt a 256-bit blob key obtained from the CAAM's RNG. This random blob key is used to encrypt the content of the blob. The encrypted blob key is stored as part of the blob. During blob decryption the BKEK is recreated so that the encrypted blob key can be decrypted, and then used to decrypt the blob content.

11.5.1 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 11-6. 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).

11.5.2 Provisioning the Zeroizable Master Key

The ZMK is programmed after or during initialization of the SNVS_LP. The ZMK can be provisioned either by software or by hardware. After the ZMK has been provisioned, the value is retained even when power is off in the most of the chip. The SNVS's LP section (SNVS_LP) includes the following registers for storing and locking the Zeroizable Master Key (ZMK).

Register	Overview
SNVS_LP Zeroizable Master Key Registers	Eight 32-bit registers store a 256-bit value called the Zeroizable Master Key (ZMK). The ZMK provides essentially the same function as the OTPMK from the Security Fuse Processor, except that the ZMK value can be zeroized if a security violation is detected. Upon detection of a security violation both the ZMK and the OTPMK are rendered unavailable during the current boot cycle, but in addition the ZMK is zeroized, which renders the ZMK value unavailable upon all subsequent boot cycles.
SNVS_LP Lock Register	This register controls read and write access to all other SNVS_LP registers. Once a bit in the SNVS_LP Lock Register is set, it cannot be cleared without a device reset.
SNVS_LP Master Key Control Register	SNVS_LP Master Key Control determines whether the ZMK, the OTPMK, or the XOR of the two, will be used by the CAAM when creating cryptographic blobs. This register also: • controls the programming mode for the ZMK (software, or hardware via CAAM's Random Number Generator) • stores the ZMK's Error Correcting Code • implements a valid bit that indicates that the ZMK has been provisioned
SNVS_LP Status Register	Provides status information about the SNVS Low Power section's security state, including hardware security violations that are directly detected within the SNVS_LP. The ZMK cannot be provisioned if the SNVS_LP is in a Fail state.

The SNVS_LP Master Key Control Register gives OEMs the option of directly writing a ZMK value to the ZMK Registers, or commanding the SNVS hardware to load a ZMK by acquiring a 256-bit random number from the CAAM's RNG. When the SNVS_LP loads its ZMK via the CAAM's RNG, hardware also calculates the associated Hamming code and loads it into the ZMK_ECC_Value field of the SNVS_LP Master Key Control Register.

To provision ZMK by software do the following:

- Verify that SNVS_LPMKCR [ZMK_HWP] is not set
- Verify that the ZMK registers are not locked for reads and writes. Verify that SNVS_HPLR [ZMK_WSL], [ZMK_RSL] or SNVS_LPLR [ZMK_WHL], [ZMK_WHL] are not set.

- Write key value to the ZMK registers
- Verify that the intended key value is written
  - Set SNVS_LPMKCR [ZMK_VAL] to indicate the ZMK is ready to be used by the CAAM.
  - (Optional but recommended) Set SNVS_LPMKCR [ZMK_ECC_EN] to enable ZMK error correction code verification. Software can verify that the correct nine-bit codeword is generated by reading ZMK_ECC_VALUE field
- (Optional but recommended) Block software read accesses to the ZMK registers and ZMK_ECC_VALUE field by setting SNVS_HPLR [ZMK_RSL] and SNVS_LPLR [ZMK_RHL]
- (Optional but recommended) Block software write accesses to the ZMK registers by setting SNVS_HPLR [ZMK_WSL] and SNVS_LPLR [ZMK_WHL]
- Set SNVS_HPCOMR [MKS_EN] and SNVS_LPMKCR [MASTER_KEY_SEL] to select combination of OTPMK and ZMK to be provided to the CAAM
- (Optional but recommended) Block software write accesses to the MASTER_KEY_SEL field by setting MKS lock bit
- (Optional but recommended) Block software write accesses to SNVS_LPMKCR [MASTER_KEY_SEL] by setting SNVS_HPLR [MKS_SL].

To provision ZMK by hardware do the following:

- Verify that the ZMK registers are not locked for writes. Check that SNVS_HPLR [ZMK_WSL], or SNVS_LPLR [ZMK_WHL] are not set
- Set SNVS_LPMKCR [ZMK_HWP]
- Set SNVS_HPCOMR [PROG_ZMK]
- Poll for SNVS_LPMKCR [ZMK_VAL] bit to be set. This bit is set by hardware at the end of the ZMK programming cycle
- (Optional but recommended) Set SNVS_LPMKCR [ZMK_ECC_EN] to enable ZMK error correction code verification by hardware. Note that the ZMK Registers and ZMK_ECC_VALUE field cannot be read by software in the hardware programming mode, meaning hardware ZMK_ECC checking is the only way to determine if the ZMK is corrupted.
- (Optional but recommended) Block hardware programming option of the ZMK Registers by setting SNVS_LPLR [ZMK_WHL]
- Set SNVS_HPCOMR [MKS_EN] and SNVS_LPMKCR [MASTER_KEY_SEL] to select combination of OTPMK and ZMK to be provided to the CAAM
- (Optional but recommended) Block software write accesses to SNVS_LPMKCR [MASTER_KEY_SEL] by setting SNVS_HPLR [MKS_SL].

If the ZMK is provisioned by hardware, the ZMK Registers and the ZMK_ECC_VALUE field in the SNVS_LP Master Key Control Register cannot be read by software.

11.6 Reset and Initialization of SNVS

SNVS is implemented in two sections (HP and LP) that both must be initialized by software. If the SNVS_LP is powered by an uninterrupted power source that is separate from main SoC power, then SNVS can operate in either of two modes, depending upon whether the main SoC power is on or off. During main SoC power-down SNVS_HP is powered-down, but 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 but the LP registers cannot be read or written. During main SoC power-up the isolation of SNVS_LP is disabled and both SNVS_HP and SNVS_LP are powered from the main SoC power. Both LP and HP registers can be read and written (locks and privilege modes permitting). Signals between the SNVS_HP and SNVS_LP sections are enabled and all SNVS functions are operational.

Since the HP and LP sections reside in different power domains, the POR for the two sections can occur at different times. If the SNVS_LP section remains powered by an uninterrupted power source when the main SoC power is off, SNVS_LP is initialized rarely, typically once when the device is first powered on and again whenever the battery is replaced. During main SoC power-up the isolation of SNVS_LP is disabled and both SNVS_HP and SNVS_LP are powered from the main SoC power. Signals between the SNVS_HP and SNVS_LP sections are enabled and all SNVS functions are operational. The SNVS_HP section is powered from the main SoC power, so it must be initialized after the device is powered on. If the SNVS_LP section is powered from the main SoC power rather than from an uninterrupted power source, the SNVS_LP section must also be initialized at SoC POR.

- Initializing the LP section
  - The following steps should be completed to properly initialize the SNVS LP section (required only on LP POR, i.e. when the battery is replaced):
    - Software should write the proper initialization value (41736166h) into the LP Digital Low-Voltage Detector Register and clear the low-voltage event record in the LP status register. See Digital Low-Voltage Detector (LVD) for more details.
    - If the SRTC will be used, set the Secure Real Time Clock.
    - If the ZMK will be used, provision the ZMK (see Provisioning the Zeroizable Master Key
    - If the Monotonic Counter will be used, burn an additional Monotonic Counter Era bit in the fuse bank and reset the Monotonic Counter.
- Initializing the HP section
  - The following steps should be completed to properly initialize the SNVS HP section (required on HP POR, i.e. SoC POR):

- Perform normal or secure boot to put the SNVS into a functional state (Non-secure, Trusted, Secure) (see HP Command Register, SSM_ST bitfield).
- Program SNVS general functions/configurations (see HP Control Register).
- Enable security violations and interrupts in the HP Security Violation Control Register and the HP Security Interrupt Control Register.
- Select Master Key (OTPMK, ZMK or XOR of the two. Default is OTPMK.)
- Set lock bits. The ms bit of the HP Lock Register should be set before starting any functional operation. Setting this bit prevents further changes to the Master Key selection.

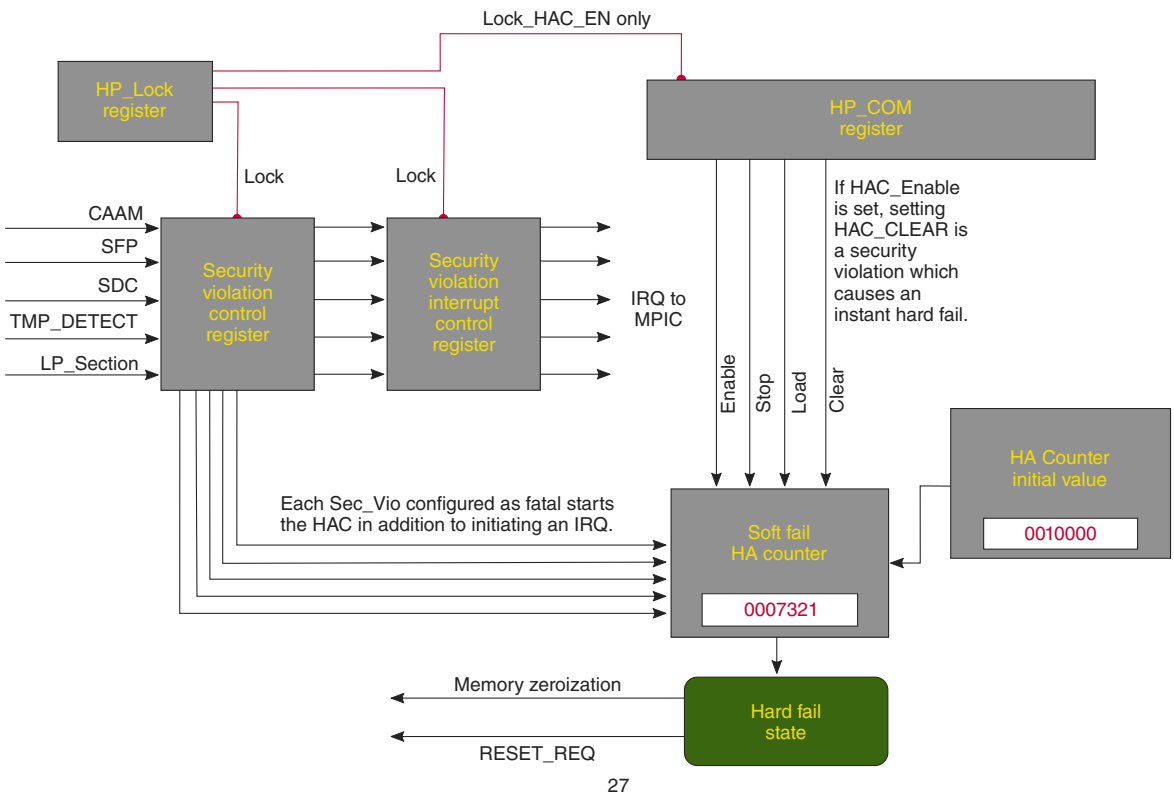


Figure 11-5. Relationship Between the Registers

11.6.1 Checklist for Initialization of the SNVS HP

At SNVS HP POR. software must perform inititialization actions in the SNVS HP section as indicated in the table below.

Initialization action (Some of these actions are performed in the boot firmware or software.)	Comments	See section
Configure fatal/nonfatal security violations	At HP POR all security violations default to disabled or nonfatal.	Configuring SNVS's response to a security event SNVS_HP Security Violation Control Register (HPSVCR)
Configure the High Assurance Counter	Can be skipped if HAC is not used. Defaults to HAC disabled at HP POR.	High Assurance Counter SNVS_HP High Assurance Counter IV Register (HPHACIVR) SNVS_HP High Assurance Counter Register (HPHACR) HAC_STOP, HAC_CLEAR, HAC_LOAD, and HAC_EN in SNVS_HP Command Register (HPCOMR) HAC_L in SNVS_HP Lock Register (HPLR)
Configure the SNVS interrupt policy	By default security violation interrupts are disabled at HP POR.	SNVS interrupts, alarms, and security violations SNVS_HP Security Interrupt Control Register (HPSICR)
Advance the Security State Machine	If software detects a security violation, force a transition to Soft Fail state, else cause a transition to Trusted state (and then to Secure state, if desired).	Security state machine SW_LPSV, SW_FSV, SW_SV, SSM_SFNS_DIS, SSM_ST_DIS and SSM_ST in SNVS_HP Command Register (HPCOMR)
Select Master Key input to CAAM	If ZMK is not used, select OTPMK (default), else select ZMK or CMK.	Configuring Master Key checking and control MASTER_KEY_ SEL in SNVS_LP Master Key Control Register (LPMKCR)

11.6.2 Checklist for Initialization of the SNVS LP

At SNVS LP POR. software must perform inititialization actions in the SNVS LP section as indicated in the table below. If the SNVS LP section is powered from the same power source as SNVS HP section, LP POR will occur at the same time as HP POR.

Initialization action (Some of these actions are performed in the boot firmware or software.)	Comments	See section
Load the Digital Low-Voltage Detector Register	Software should load the register with the value 4173_6166h and then clear the digital low-voltage detector status bit in the LP Status Register.	Digital Low-Voltage Detector (LVD)
Burn additional MC_ERA fuse and initialize Monotonic Counter	Can be skipped if Monotonic Counter is unused. Should be skipped if SNVS LP is unpowered when main chip power is off.	Using the Monotonic Counter (MC) SNVS_LP Secure Monotonic Counter MSB Register (LPSMCMR) SNVS_LP Secure Monotonic Counter LSB Register (LPSMCLR) MC_SL in SNVS_HP Lock Register (HPLR) MC_HL in SNVS_LP Lock Register (LPLR) MC_ENV in SNVS_LP Control Register (LPCR)
Initialize Secure Real Time Counter	Can be skipped if Secure Real Time Counter is unused. Should be skipped if SNVS LP is unpowered when main chip power is off.	SNVS_LP Secure Real Time Counter (SRTC) SNVS_LP Secure Real Time Counter MSB Register (LPSRTCMR) SNVS_LP Secure Real Time Counter LSB Register (LPSRTCLR) SNVS_LP Time Alarm Register (LPTAR) SNVS_LP Time Alarm Register (LPTAR) LPCALB_VAL, LPCALB_EN, SRTC_INV_EN, LPTA_EN, SRTC_ENV in SNVS_LP Control Register (LPCR)
Program the Zeroizable Master Key	Can be skipped if Zeroizable Master Key is unused. Should be skipped if SNVS LP is unpowered when main chip power is off.	Provisioning the Zeroizable Master Key SNVS_LP Zeroizable Master Key Register (LPZMKR0 - LPZMKR7) SNVS_LP Master Key Control Register (LPMKCR) ZMK_RSL and ZMK_WSL in SNVS_HP Lock Register (HPLR) PROG_ZMK in SNVS_HP Command Register (HPCOMR) ZMK_ZERO in SNVS_HP Status Register (HPSR) ZMK_ECC_FAIL and ZMK_SYNDROME in SNVS_HP Security Violation Status Register (HPSVSR)

11.7 SNVS register descriptions

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 functions, 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 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.

The following table shows the SNVS main memory map.

11.7.1 SNVS memory map

SNVS base address: 3037_0000h

Offset (hex)	Register	Width (In bits)	Access	Reset value (hex)
0	SNVS_HP Lock Register (HPLR)	32	RW	0000_0000
4	SNVS_HP Command Register (HPCOMR)	32	RW	0000_0000
8	SNVS_HP Control Register (HPCR)	32	RW	0000_0000
C	SNVS_HP Security Interrupt Control Register (HPSICR)	32	RW	0000_0000
10	SNVS_HP Security Violation Control Register (HPSVCR)	32	RW	0000_0000
14	SNVS_HP Status Register (HPSR)	32	W1C	8000_B000
18	SNVS_HP Security Violation Status Register (HPSVSR)	32	W1C	8000_0000
1C	SNVS_HP High Assurance Counter IV Register (HPHACIVR)	32	RW	0000_0000
20	SNVS_HP High Assurance Counter Register (HPHACR)	32	RO	0000_0000
24	SNVS_HP Real Time Counter MSB Register (HPRTCMR)	32	RW	0000_0000
28	SNVS_HP Real Time Counter LSB Register (HPRTCLR)	32	RW	0000_0000
2C	SNVS_HP Time Alarm MSB Register (HPTAMR)	32	RW	0000_0000
30	SNVS_HP Time Alarm LSB Register (HPTALR)	32	RW	0000_0000
34	SNVS_LP Lock Register (LPLR)	32	RW	0000_0000
38	SNVS_LP Control Register (LPCR)	32	RW	0000_0020
3C	SNVS_LP Master Key Control Register (LPMKCR)	32	RW	0000_0000
40	SNVS_LP Security Violation Control Register (LPSVCR)	32	RW	0000_0000
48	SNVS_LP Security Events Configuration Register (LPSECR)	32	RW	0000_0000
4C	SNVS_LP Status Register (LPSR)	32	RW	0000_0008
50	SNVS_LP Secure Real Time Counter MSB Register (LPSRTCMR)	32	RW	0000_0000
54	SNVS_LP Secure Real Time Counter LSB Register (LPSRTCLR)	32	RW	0000_0000
58	SNVS_LP Time Alarm Register (LPTAR)	32	RW	0000_0000
5C	SNVS_LP Secure Monotonic Counter MSB Register (LPSMCMR)	32	RW	0000_0000
60	SNVS_LP Secure Monotonic Counter LSB Register (LPSMCLR)	32	RW	0000_0000
64	SNVS_LP Digital Low-Voltage Detector Register (LPLVDR)	32	RW	0000_0000
68	SNVS_LP General Purpose Register 0 (legacy alias) (LPGPR0_legacy_alias)	32	RW	0000_0000
6C - 88	SNVS_LP Zeroizable Master Key Register (LPZMKR0 - LPZMKR7)	32	RW	0000_0000
90 - 9C	SNVS_LP General Purpose Registers 0 .. 3 (LPGPR0 - LPGPR3)	32	RW	0000_0000
BF8	SNVS_HP Version ID Register 1 (HPVIDR1)	32	RO	003E_0103
BFC	SNVS_HP Version ID Register 2 (HPVIDR2)	32	RO	0600_0300

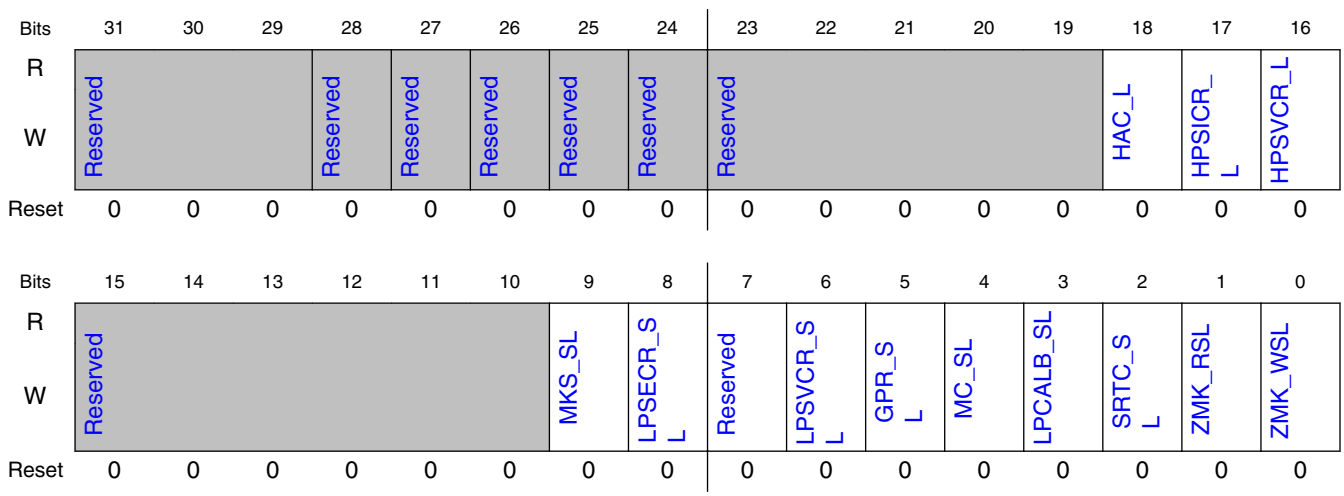
11.7.2 SNVS_HP Lock Register (HPLR)

The SNVS_HP Lock Register contains lock bits for the SNVS registers. This is a privileged write register.

11.7.2.1 Offset

Register	Offset
HPLR	0h

11.7.2.2 Diagram



11.7.2.3 Fields

Field	Description
31-29 —	Reserved
28 —	Reserved
27 —	Reserved

Table continues on the next page...

Field	Description
26 —	Reserved
25 —	Reserved
24 —	Reserved
23-19 —	Reserved
18 HAC_L	High Assurance Counter 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 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 —	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 LPSECR_SL	LP Security Events Configuration Register Soft Lock When set, prevents any writes to the LPSECR. Once set, this bit can only be reset by the system reset. 0 - Write access is allowed 1 - Write access is not allowed
7 —	Reserved
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.

Table continues on the next page...

Field	Description
	0 - Write access is allowed 1 - Write access is not allowed
4 MC_SL	Monotonic Counter Soft 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 system reset. 0 - Write access (increment) is allowed 1 - Write access (increment) is not allowed
3 LPCALB_SL	LP Calibration Soft 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 system reset. 0 - Write access is allowed 1 - Write access is not allowed
2 SRTC_SL	Secure Real Time Counter Soft 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 system reset. 0 - Write access is allowed 1 - Write access is not allowed
1 ZMK_RSL	Zeroizable Master Key Read Soft 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), 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 system reset. 0 - Read access is allowed (only in software Programming mode) 1 - Read access is not allowed
0 ZMK_WSL	Zeroizable Master Key Write Soft Lock When set, prevents any writes (software and hardware) to the ZMK registers and the ZMK_HWP, ZMK_VAL, and ZMK_ECC_EN fields of the LPMKCR. Once set, this bit can only be cleared by system reset. 0 - Write access is allowed 1 - Write access is not allowed

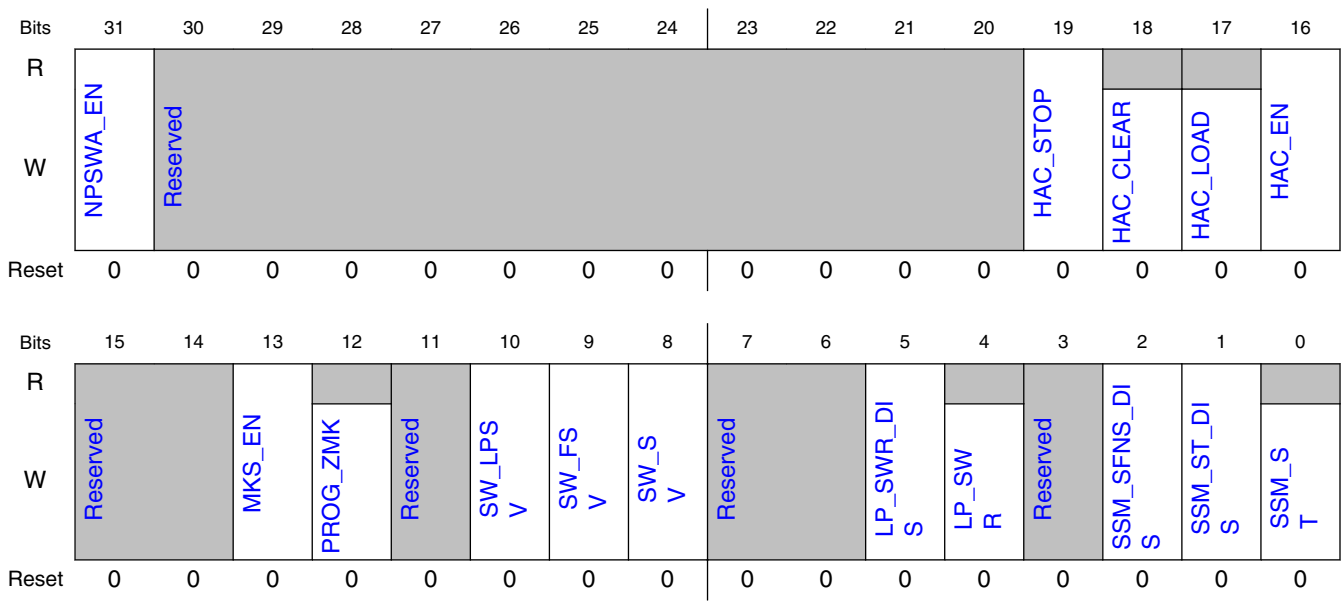
11.7.3 SNVS_HP Command Register (HPCOMR)

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.

11.7.3.1 Offset

Register	Offset
HPCOMR	4h

11.7.3.2 Diagram



11.7.3.3 Fields

Field	Description
31 NPSWA_EN	Non-Privileged Software Access Enable When set, allows non-privileged software to access all SNVS registers, including those that are privileged software read/write access only. 0 Only privileged software can access privileged registers 1 Any software can access privileged registers
30-20 —	Reserved
19 HAC_STOP	High Assurance Counter Stop 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.

Table continues on the next page...

Field	Description
	0 HAC counter can count down 1 HAC counter is stopped
18 HAC_CLEAR	High Assurance Counter Clear 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 counter is enabled (HAC_EN is set). This self-clearing bit is always read as zero. 0 - No Action 1 - Clear the HAC
17 HAC_LOAD	High Assurance Counter Load 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. 0 - No Action 1 - Load the HAC
16 HAC_EN	High Assurance Counter Enable 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. 0 - High Assurance Counter is disabled 1 - High Assurance Counter is enabled
15-14 —	Reserved
13 MKS_EN	Master Key Select Enable 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. 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
12 PROG_ZMK	Program Zeroizable Master Key This bit activates ZMK hardware programming mechanism. This mechanism is activated only if the ZMK is configured to the hardware programming mode and ZMK in not locked for writes. This self-clearing bit is always read as zero. 0 - No Action 1 - Activate hardware key programming mechanism
11 —	Reserved
10 SW_LPSV	LP Software Security Violation 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.
9 SW_FSV	Software Fatal Security Violation 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: Check State -> Soft Fail

Table continues on the next page...

Field	Description
	Non-Secure State -> Soft Fail Trusted State -> Soft Fail Secure State -> Soft Fail
8 SW_SV	Software Security Violation 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: Check -> Non-Secure Trusted -> Soft Fail Secure -> Soft Fail
7-6 —	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 to 1, most registers in the SNVS_LP section are reset, but the following registers are *not* reset by an LP software reset: • Secure Real Time Counter • Time Alarm Register 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 —	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)

Field	Description
	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)

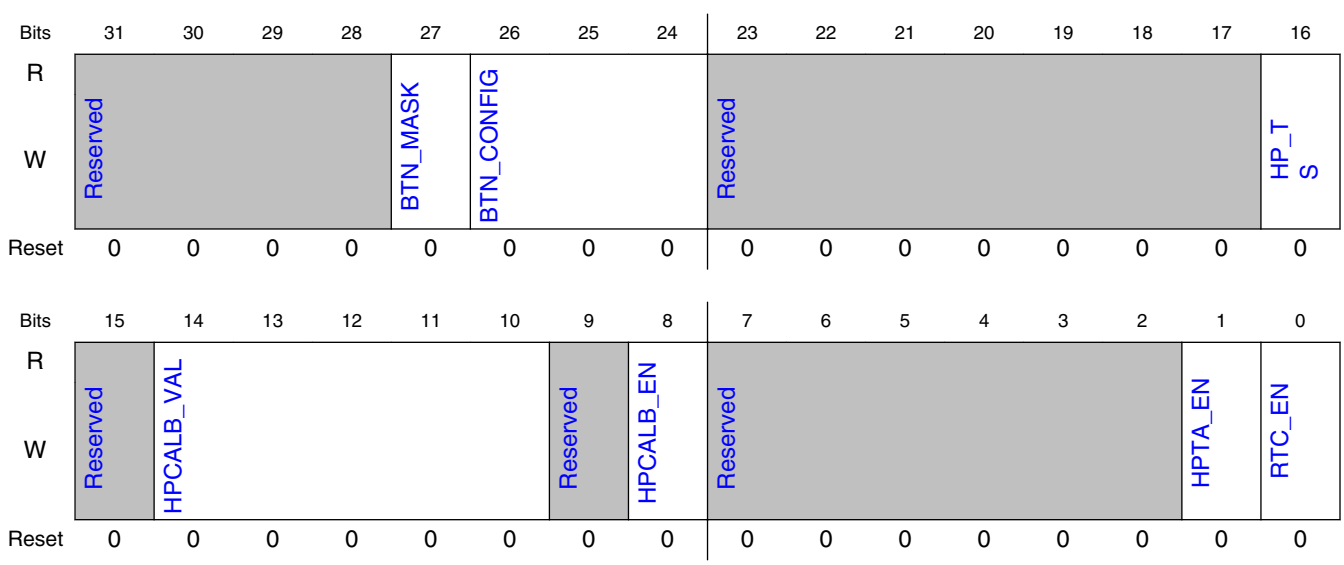
11.7.4 SNVS_HP Control Register (HPCR)

The SNVS_HP Control Register contains various control bits of the HP section of SNVS. This is *not* a privileged write register.

11.7.4.1 Offset

Register	Offset
HPCR	8h

11.7.4.2 Diagram



11.7.4.3 Fields

Field	Description
31-28 —	Reserved
27 BTN_MASK	Button interrupt mask. This bit is used to mask the button (BTN) interrupt request. 0: Interrupt disabled 1: Interrupt enabled
26-24 BTN_CONFIG	Button Configuration. This field is used to configure which feature of the button (BTN) input signal constitutes "active". 000: Button signal is active high 001: Button signal is active low 010: Button signal is active on the falling edge 011: Button signal is active on the rising edge 100: Button signal is active on any edge All other patterns are Reserved
23-17 —	Reserved
16 HP_TS	HP Time Synchronize. HP RTC should be disabled when synchronizing counter value. When set, this updates the HP Real Time Counter with the LP Real 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 —	Reserved
14-10 HPCALB_VAL	HP Calibration Value 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 —	Reserved

Table continues on the next page...

Field	Description
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
7-2 —	Reserved
1 HPTA_EN	HP Time Alarm Enable 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. 0 - HP Time Alarm Interrupt is disabled 1 - HP Time Alarm Interrupt is enabled
0 RTC_EN	HP Real Time Counter Enable. This bit syncs with the 32KHz clock. It won't update with the bus clock. 0 - RTC is disabled 1 - RTC is enabled

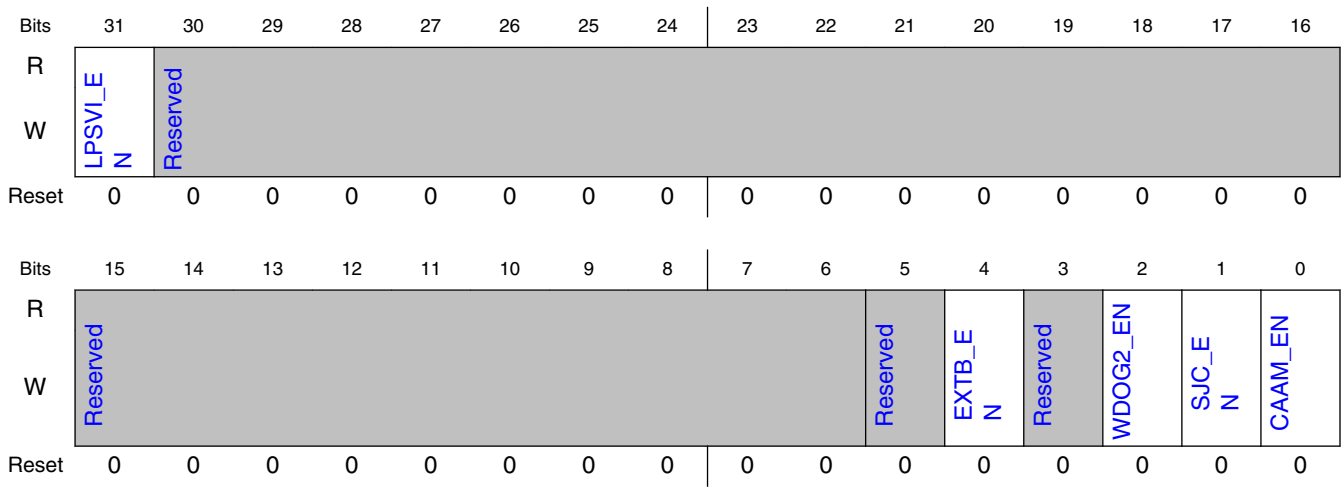
11.7.5 SNVS_HP Security Interrupt Control Register (HPSICR)

The HP Security Interrupt Control Register defines the SNVS security interrupt generation policy. This is a privileged write register.

11.7.5.1 Offset

Register	Offset
HPSICR	Ch

11.7.5.2 Diagram



11.7.5.3 Fields

Field	Description
31 LPSVI_EN	LP Security Violation Interrupt Enable This bit enables generating of the security interrupt to the host processor upon security violation signal from the LP section. 0 - LP Security Violation Interrupt is Disabled 1 - LP Security Violation Interrupt is Enabled
30-6 —	Reserved
5 —	Reserved
4 EXTB_EN	External Boot Interrupt Enable Setting this bit to 1 enables generation of the security interrupt to the host processor upon detection of the External Boot security violation. 0 - External Boot Interrupt is Disabled 1 - External Boot Interrupt is Enabled
3 —	Reserved
2 WDOG2_EN	Watchdog 2 reset Interrupt Enable Setting this bit to 1 enables generation of the security interrupt to the host processor upon detection of the Watchdog 2 reset security violation. 0 - Watchdog 2 reset Interrupt is Disabled 1 - Watchdog 2 reset Interrupt is Enabled
1	JTAG Active Interrupt Enable

Table continues on the next page...

Field	Description
SJC_EN	Setting this bit to 1 enables generation of the security interrupt to the host processor upon detection of the JTAG Active security violation. 0 - JTAG Active Interrupt is Disabled 1 - JTAG Active Interrupt is Enabled
0 CAAM_EN	CAAM Security Violation Interrupt Enable Setting this bit to 1 enables generation of the security interrupt to the host processor upon detection of the CAAM Security Violation security violation. 0 - CAAM Security Violation Interrupt is Disabled 1 - CAAM Security Violation Interrupt is Enabled

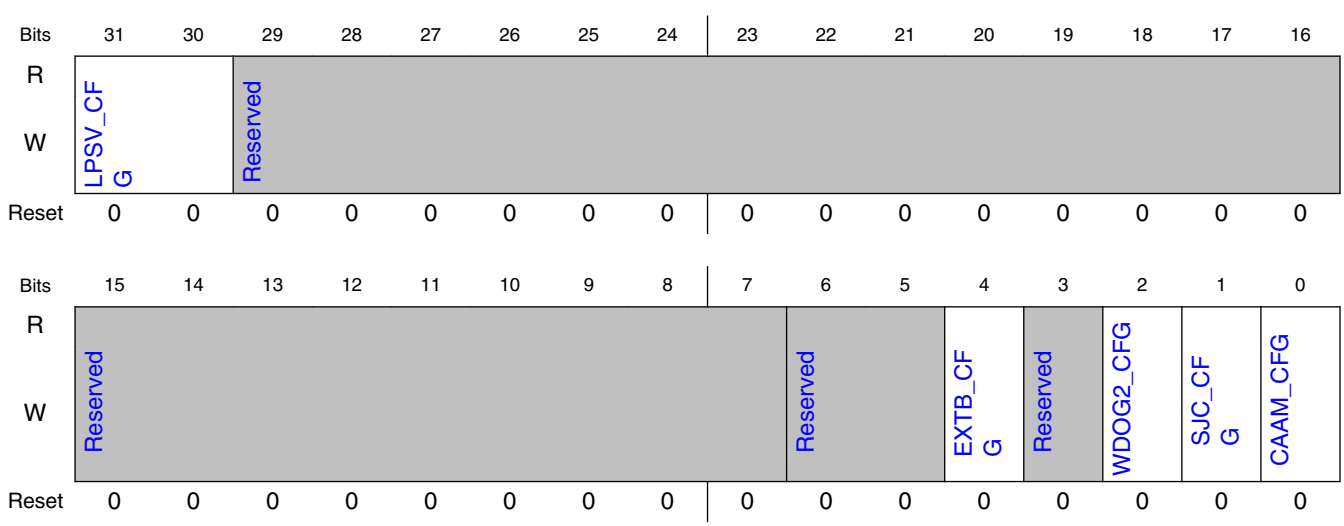
11.7.6 SNVS_HP Security Violation Control Register (HPSVCR)

The HP Security Violation Control Register defines types for each security violation input. This is a privileged write register.

11.7.6.1 Offset

Register	Offset
HPSVCR	10h

11.7.6.2 Diagram



11.7.6.3 Fields

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 —	Reserved
6-5 —	Reserved
4 EXTB_CFG	External Boot Security Violation Configuration This field configures the External Boot Security Violation Input. This setting instructs the SSM how to respond upon detection of the External Boot security violation. 0 - External Boot is a non-fatal violation 1 - External Boot is a fatal violation
3 —	Reserved
2 WDOG2_CFG	Watchdog 2 reset Security Violation Configuration This field configures the Watchdog 2 reset Security Violation Input. This setting instructs the SSM how to respond upon detection of the Watchdog 2 reset security violation. 0 - Watchdog 2 reset is a non-fatal violation 1 - Watchdog 2 reset is a fatal violation
1 SJC_CFG	JTAG Active Security Violation Configuration This field configures the JTAG Active Security Violation Input. This setting instructs the SSM how to respond upon detection of the JTAG Active security violation. 0 - JTAG Active is a non-fatal violation 1 - JTAG Active is a fatal violation
0 CAAM_CFG	CAAM Security Violation Security Violation Configuration This field configures the CAAM Security Violation Security Violation Input. This setting instructs the SSM how to respond upon detection of the CAAM Security Violation security violation. 0 - CAAM Security Violation is a non-fatal violation 1 - CAAM Security Violation is a fatal violation

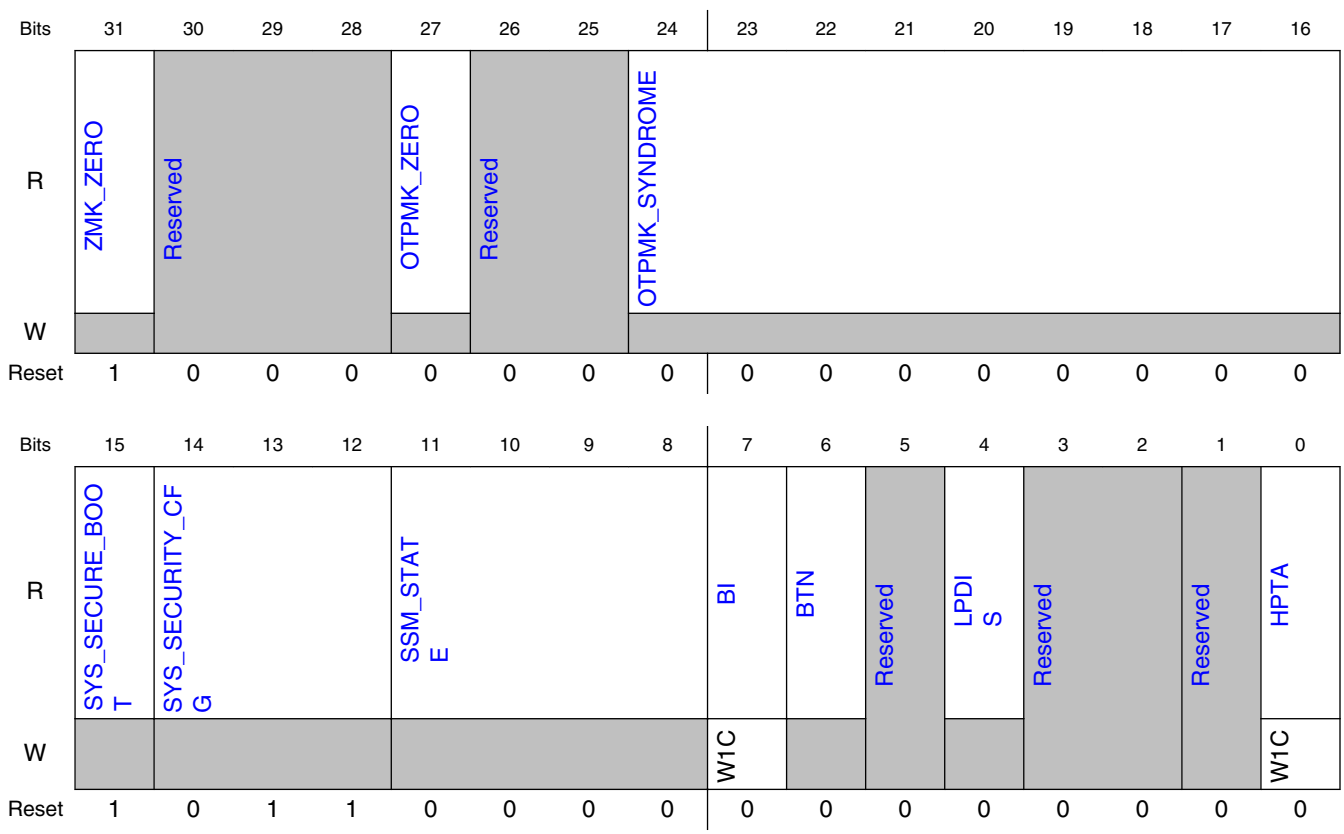
11.7.7 SNVS_HP Status Register (HPSR)

The HP Status Register reflects the internal state of the SNVS. This is *not* a privileged write register.

11.7.7.1 Offset

Register	Offset
HPSR	14h

11.7.7.2 Diagram



11.7.7.3 Fields

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. This bit will reset to 1 on LP Section POR. If the LP section is powered by an uninterrupted power source, once the ZMK has been programmed with a nonzero value this bit will reset to 0 on HP Section PORs.

Table continues on the next page...

Field	Description
	0 - The ZMK is not zero. 1 - The ZMK is zero.
30-28 —	Reserved
27 OTPMK_ZERO	One Time Programmable Master Key is Equal to Zero. When set, this bit always triggers "bad key" violation. OTPMK_ZERO will normally reset to 0. This will reset to 1 only if the OTPMK has not been programmed in the fuse bank. 0 - The OTPMK is not zero. 1 - The OTPMK is zero.
26-25 —	Reserved
24-16 OTPMK_SYNDROME	One Time Programmable Master Key Syndrome In the case of a single-bit error, the eight lower bits of this value indicate the bit number of error location. For example, syndrome word 10010110 indicates that key bit 150 (lsb=0) has an error. The ninth bit of the syndrome word checks parity of the whole key value. This bit is 1 when an odd number of errors has 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. OTPMK_SYNDROME will normally reset to 0x000. This will reset to nonzero only if the OTPMK has an error in one or more bits.
15 SYS_SECURE_BOOT	System Secure Boot If SYS_SECURE_BOOT is 1, the chip boots from internal ROM. In a chip in the field, SYS_SECURE_BOOT will normally reset to 1. It will reset to 0 only in a test chip.
14-12 SYS_SECURITY_CFG	System Security Configuration This field reflects the three security configuration inputs to SNVS. These inputs are used in conjunction with the sys_secure_boot input, which is visible as the SYS_SECURE_BOOT bit. 000 - Fab Configuration - the default configuration of newly fabricated chips 001 - Open Configuration - the configuration after NXP-programmable fuses have been blown 011 - Closed Configuration - the configuration after OEM-programmable fuses have been blown 111 - Field Return Configuration - the configuration of chips that are returned to NXP for analysis
11-8 SSM_STATE	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 0001 - Hard Fail 0011 - Soft Fail 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

Table continues on the next page...

Field	Description
	1111 - Secure
7 BI	Button Interrupt Signal ipi_snvs_btn_int_b was asserted.
6 BTN	Button Value of the BTN input. This is the external button used for PMIC control. 0: BTN not pressed 1: BTN pressed
5 —	Reserved
4 LPDIS	Low Power Disable If 1, the low power section has been disabled by means of an input signal to SNVS.
3-2 —	Reserved
1 —	Reserved
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.

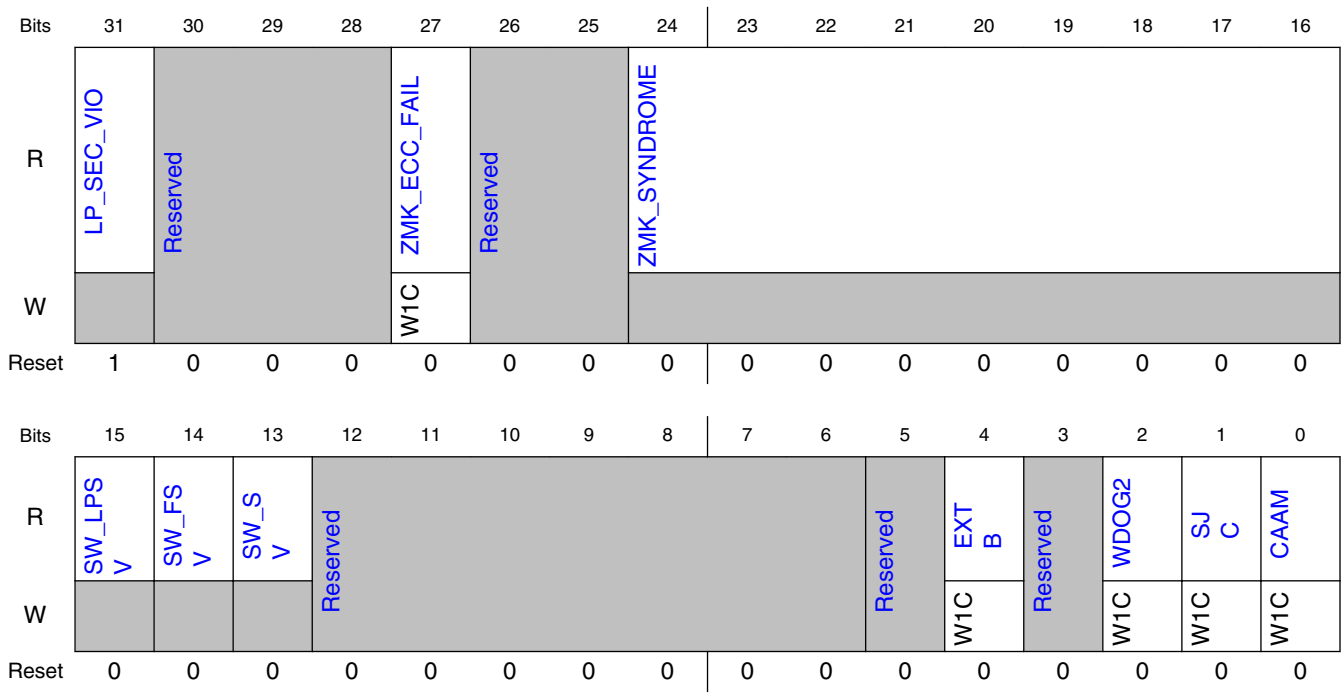
11.7.8 SNVS_HP Security Violation Status Register (HPSVSR)

The HP Security Violation Status Register reflects the HP domain security violation records. Write a 1 to Security Violation 5 (reserved)-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. This is a privileged write register.

11.7.8.1 Offset

Register	Offset
HPSVSR	18h

11.7.8.2 Diagram



11.7.8.3 Fields

Field	Description
31 LP_SEC_VIO	LP Security Violation A security volation was detected in the SNVS low power section. This bit will reset to 1 on LP Section POR because the Digital Low-Voltage Detector will indicate that an LP section low-voltage event has occurred. If the LP section is powered by an uninterrupted power source, LP Security Violation will normally reset to 0 if no LP POR has occurred since the Digital Low-Voltage Detector was written. If a POR occurred in the LP section, at HP POR the boot ROM firmware will normally detect this and will then re-initialize the LP section and reset LP Security Violation to 0.
30-28 —	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 and also clears this register's ZMK_SYNDROME field. 0 - ZMK ECC Failure was not detected. 1 - ZMK ECC Failure was detected.
26-25 —	Reserved
24-16	Zeroizable Master Key Syndrome

Table continues on the next page...

Field	Description
ZMK_SYNDROME	The ZMK syndrome indicates the single-bit error location and parity for the ZMK register. It operates 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 SW_LPSV	LP Software Security Violation This bit is a read-only copy of the SW_LPSV bit in the HP Command Register.
14 SW_FSV	Software Fatal Security Violation This bit is a read-only copy of the SW_FSV bit in the HP Command Register.
13 SW_SV	Software Security Violation This bit is a read-only copy of the SW_SV bit in the HP Command Register.
12-6 —	Reserved
5 —	Reserved
4 EXTB	External Boot security violation was detected. 0 - No External Boot security violation was detected. 1 - External Boot security violation was detected.
3 —	Reserved
2 WDOG2	Watchdog 2 reset security violation was detected. 0 - No Watchdog 2 reset security violation was detected. 1 - Watchdog 2 reset security violation was detected.
1 SJC	JTAG Active security violation was detected. 0 - No JTAG Active security violation was detected. 1 - JTAG Active security violation was detected.
0 CAAM	CAAM Security Violation security violation was detected. 0 - No CAAM Security Violation security violation was detected. 1 - CAAM Security Violation security violation was detected.

11.7.9 SNVS_HP High Assurance Counter IV Register (HPHACIVR)

The SNVS_HP High Assurance Counter IV Register contains the initial value for the high assurance counter. This is a privileged write register.

11.7.9.1 Offset

Register	Offset
HPHACIVR	1Ch

11.7.9.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	HAC_COUNTER_IV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	HAC_COUNTER_IV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.9.3 Fields

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.

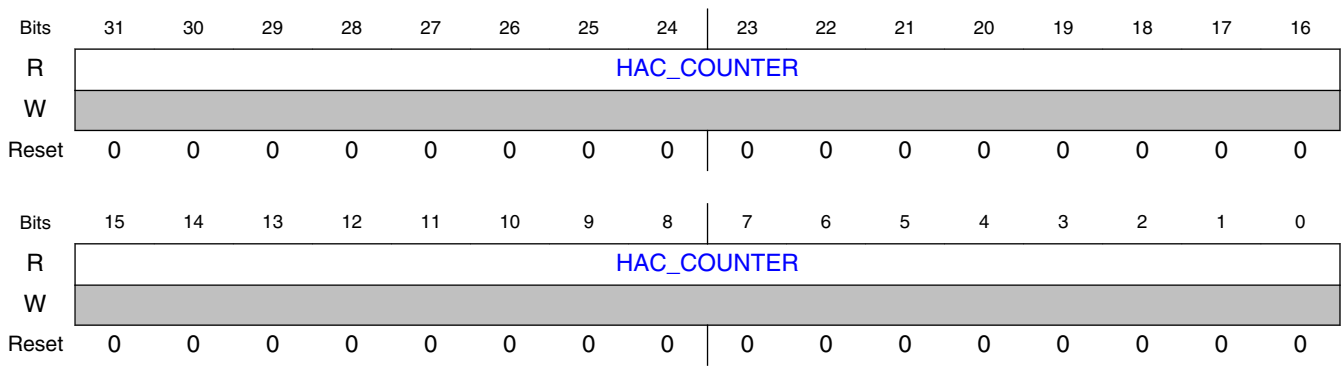
11.7.10 SNVS_HP High Assurance Counter Register (HPHACR)

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. This is a non-privileged read-only register.

11.7.10.1 Offset

Register	Offset
HPHACR	20h

11.7.10.2 Diagram



11.7.10.3 Fields

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. • When HAC_STOP bit is set, the HAC Counter is stopped. • When HAC_CLEAR bit is set, the HAC Counter is cleared. • When HAC_LOAD bit is set, the HAC Counter is loaded with the value of the HPHACIVR.

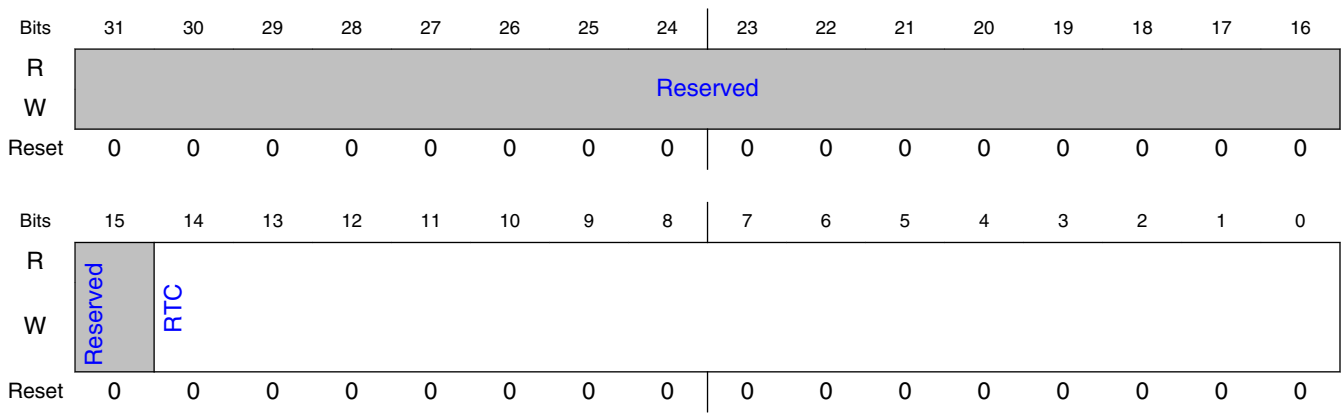
11.7.11 SNVS_HP Real Time Counter MSB Register (HPRTCMR)

The SNVS_HP Real Time Counter MSB register contains the 15 most-significant bits of the HP Real Time Counter. This is *not* a privileged write register.

11.7.11.1 Offset

Register	Offset
HPRTCMR	24h

11.7.11.2 Diagram



11.7.11.3 Fields

Field	Description
31-15 —	Reserved
14-0 RTC	HP Real Time Counter The most-significant 15 bits of the RTC. This register can be programmed only when RTC is not active (RTC_EN bit is not set).

11.7.12 SNVS_HP Real Time Counter LSB Register (HPRTCLR)

The SNVS_HP Real Time Counter LSB register contains the 32 least-significant bits of the HP real time counter. This is *not* a privileged write register.

11.7.12.1 Offset

Register	Offset
HPRTCLR	28h

11.7.12.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RTC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RTC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.12.3 Fields

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).

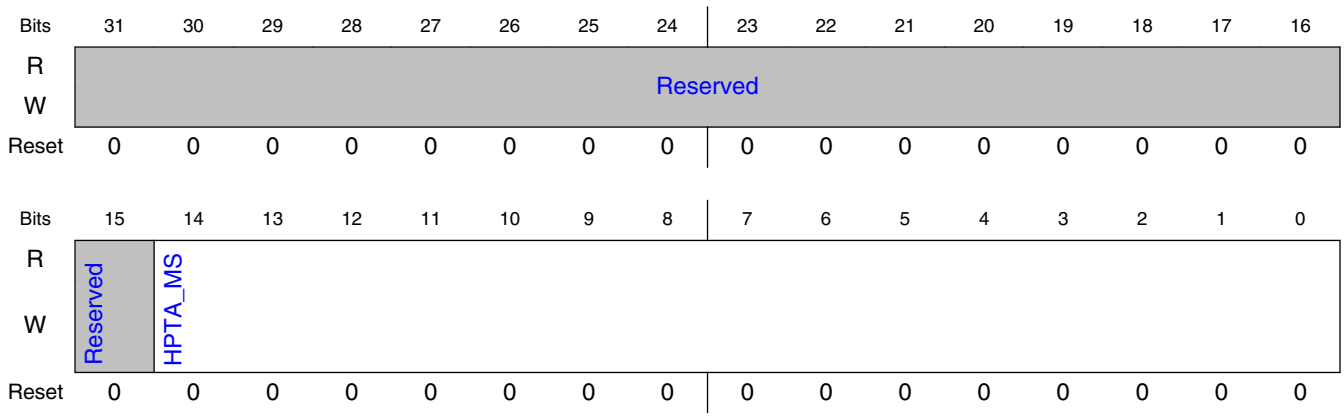
11.7.13 SNVS_HP Time Alarm MSB Register (HPTAMR)

The SNVS_HP Time Alarm MSB register contains the most-significant bits of the SNVS_HP Time Alarm value. This is *not* a privileged write register.

11.7.13.1 Offset

Register	Offset
HPTAMR	2Ch

11.7.13.2 Diagram



11.7.13.3 Fields

Field	Description
31-15 —	Reserved
14-0 HPTA_MS	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).

11.7.14 SNVS_HP Time Alarm LSB Register (HPTALR)

The SNVS_HP Time Alarm LSB register contains the 32 least-significant bits of the SNVS_HP Time Alarm value. This is *not* a privileged write register.

11.7.14.1 Offset

Register	Offset
HPTALR	30h

11.7.14.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	HPTA_LS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	HPTA_LS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.14.3 Fields

Field	Description
31-0 HPTA_LS	HP Time Alarm, 32 least-significant bits. This register can be programmed only when HP time alarm is disabled (HPTA_EN bit is not set).

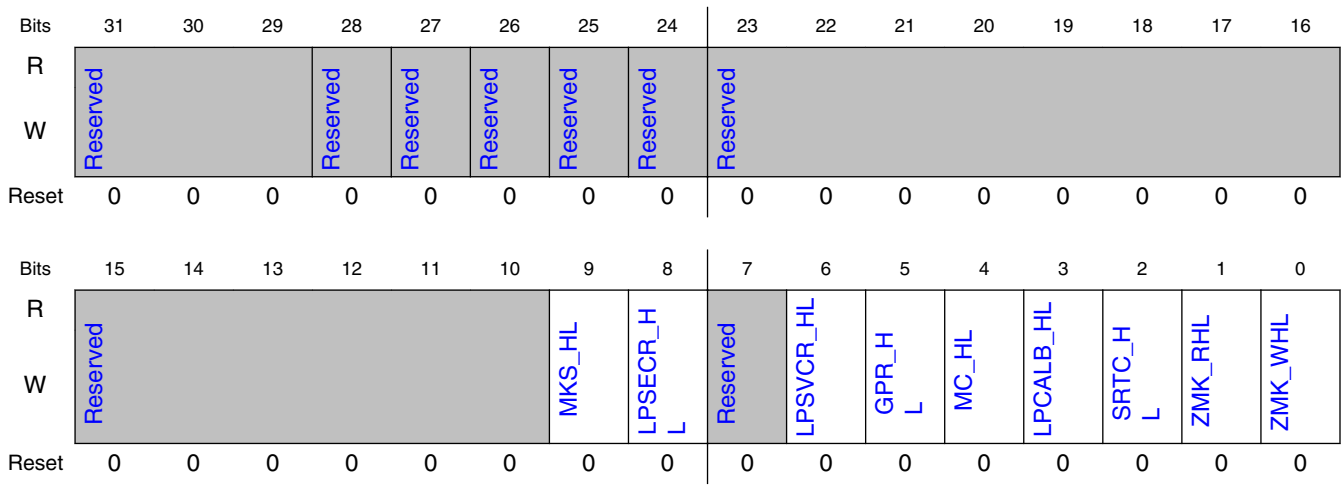
11.7.15 SNVS_LP Lock Register (LPLR)

The SNVS_LP Lock Register contains lock bits for the SNVS_LP registers. This is a privileged write register.

11.7.15.1 Offset

Register	Offset
LPLR	34h

11.7.15.2 Diagram



11.7.15.3 Fields

Field	Description
31-29 —	Reserved
28 —	Reserved
27 —	Reserved
26 —	Reserved
25 —	Reserved
24 —	Reserved
23-10 —	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 LPSECR_HL	LP Security Events Configuration Register Hard Lock When set, prevents any writes to the LPSECR. Once set, this bit can only be reset by the LP POR.

Table continues on the next page...

Field	Description
	0 - Write access is allowed. 1 - Write access is not allowed.
7 —	Reserved
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. 0 - Write access is allowed. 1 - Write access is not allowed.
5 GPR_HL	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. 0 - Write access is allowed. 1 - Write access is not allowed.
4 MC_HL	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. 0 - Write access (increment) is allowed. 1 - Write access (increment) is not allowed.
3 LPCALB_HL	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. 0 - Write access is allowed. 1 - Write access is not allowed.
2 SRTC_HL	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. 0 - Write access is allowed. 1 - Write access is not allowed.
1 ZMK_RHL	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. 0 - Read access is allowed (only in software programming mode). 1 - Read access is not allowed.
0 ZMK_WHL	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. 0 - Write access is allowed. 1 - Write access is not allowed.

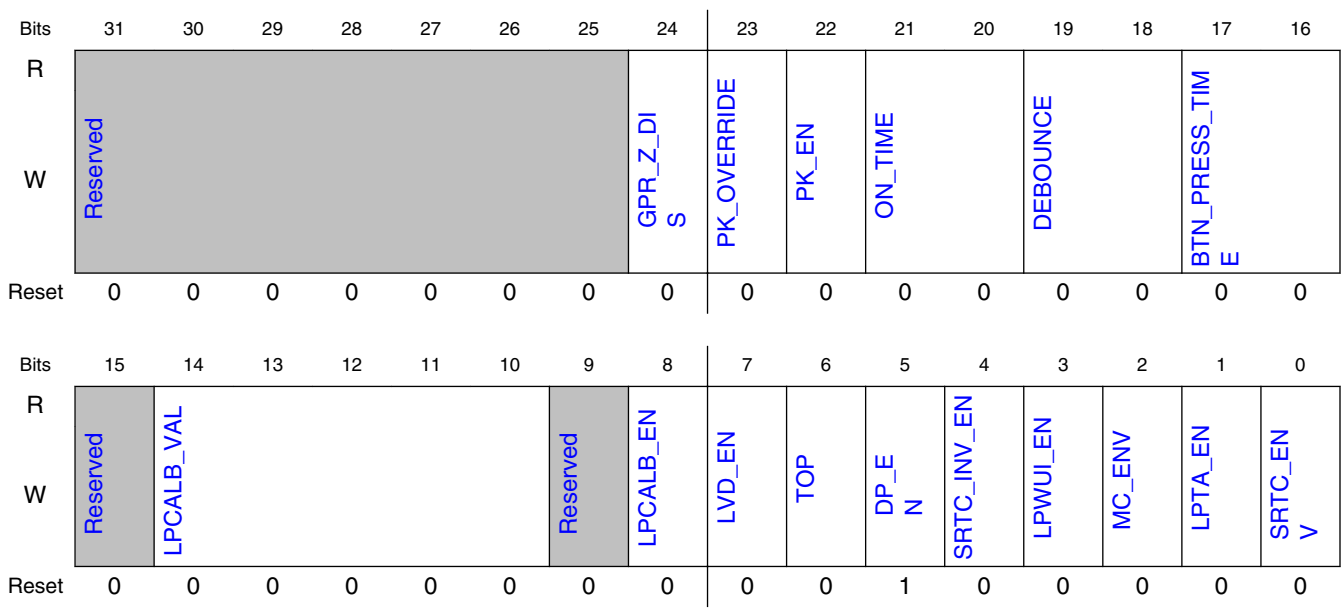
11.7.16 SNVS_LP Control Register (LPCR)

The SNVS_LP Control Register contains various control bits of the LP section of SNVS. This is a privileged write register.

11.7.16.1 Offset

Register	Offset
LPCR	38h

11.7.16.2 Diagram



11.7.16.3 Fields

Field	Description
31-25 —	Reserved
24 GPR_Z_DIS	General Purpose Registers Zeroization Disable. 1 = Disable zeroization of the GPR registers when a security event occurs.

Table continues on the next page...

Field	Description
	0 = Zeroize the GPR registers when a security event occurs. (Default)
23 PK_OVERRIDE	PMIC On Request Override The value written to PK_OVERRIDE will be asserted on output signal snvs_lp_pk_override. That signal is used to override the IOMUX control for the PMIC I/O pad.
22 PK_EN	PMIC On Request Enable The value written to PK_EN will be asserted on output signal snvs_lp_pk_en. That signal is used to turn off the pullup/pulldown circuitry in the PMIC I/O pad.
21-20 ON_TIME	The ON_TIME field is used to configure the period of time after BTN is asserted before pmic_en_b is asserted to turn on the SoC power. 00: 500msec off->on transition time 01: 50msec off->on transition time 10: 100msec off->on transition time 11: 0msec off->on transition time
19-18 DEBOUNCE	This field configures the amount of debounce time for the BTN input signal. 00: 50msec debounce 01: 100msec debounce 10: 500msec debounce 11: 0msec debounce
17-16 BTN_PRESS_TIME	This field configures the button press time out values for the PMIC Logic. 00 : 5 secs 01 : 10 secs 10 : 15 secs 11 : long press disabled (pmic_en_b will not be asserted regardlessof how long BTN is asserted)
15 —	Reserved
14-10 LPCALB_VAL	LP Calibration Value 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 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 11110 - -2 counts per each 32768 ticks of the counter clock 11111 - -1 counts per each 32768 ticks of the counter clock
9 —	Reserved
8	LP Calibration Enable

Table continues on the next page...

Field	Description
LPCALB_EN	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 LVD_EN	Digital Low-Voltage Event Enable By default the detection of a low-voltage event does not cause the pmic_en_b signal to be asserted. Setting the Digital Low-Voltage Event Enable bit to 1 enables the low-voltage event for the PMIC. 0 - disabled 1 - enabled
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	If this bit is 1, in the case of a security violation the SRTC stops counting and the SRTC is invalidated (SRTC_ENV bit is cleared). This is intended to allow software to read the time at which the security violation occurred. This field cannot be changed once SRTC_SL or SRTC_HL bit is set. Software security violations stop the SRTC and are unaffected by this field. 0 - SRTC stays valid in the case of security violation (other than a software violation (HPSVSR[SW_LPSV] = 1 or HPCOMR[SW_LPSV] = 1)). 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 (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 Enabled and Valid When set, the MC can be incremented (by write transaction to the LPSMCMR or LPSMCLR). Once MC_SL or MC_HL bit is set this bit can be changed only by LP software reset or LP POR. 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. 0 - LP time alarm interrupt is disabled. 1 - LP time alarm interrupt is enabled.
0	Secure Real Time Counter Enabled and Valid

Field	Description
SRTC_ENV	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.

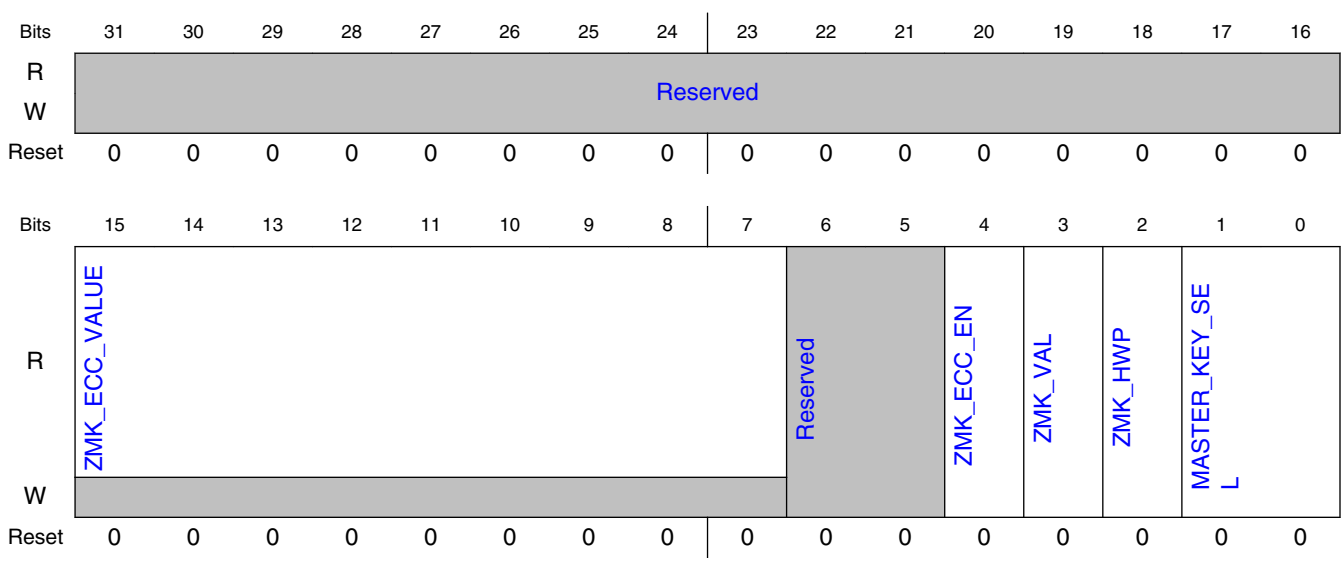
11.7.17 SNVS_LP Master Key Control Register (LPMKCR)

The SNVS_LP Master Key Control Register contains the master keys configuration. This is a privileged write register.

11.7.17.1 Offset

Register	Offset
LPMKCR	3Ch

11.7.17.2 Diagram



11.7.17.3 Fields

Field	Description
31-16 —	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. Read restrictions similar to the ZMK Registers are applied to this field (see Provisioning the Zeroizable Master Key).
6-5 —	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 Provisioning the Zeroizable Master Key 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 the 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 .

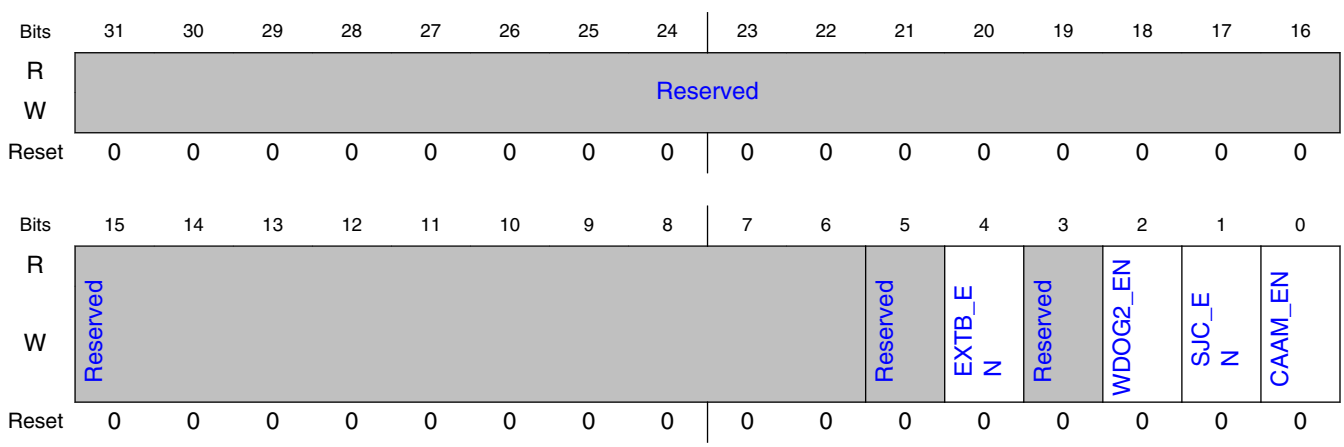
11.7.18 SNVS_LP Security Violation Control Register (LPSVCR)

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. This is a privileged write register.

11.7.18.1 Offset

Register	Offset
LPSVCR	40h

11.7.18.2 Diagram



11.7.18.3 Fields

Field	Description
31-6 —	Reserved
5	Reserved

Table continues on the next page...

Field	Description
—	
4 EXTB_EN	External Boot Enable This bit enables External Boot Input. When set, a External Boot causes an LP security violation, which clears LP sensitive data. 0 - External Boot is disabled in the LP domain. 1 - External Boot is enabled in the LP domain.
3 —	Reserved
2 WDOG2_EN	Watchdog 2 reset Enable This bit enables Watchdog 2 reset Input. When set, a Watchdog 2 reset causes an LP security violation, which clears LP sensitive data. 0 - Watchdog 2 reset is disabled in the LP domain. 1 - Watchdog 2 reset is enabled in the LP domain.
1 SJC_EN	JTAG Active Enable This bit enables JTAG Active Input. When set, a JTAG Active causes an LP security violation, which clears LP sensitive data. 0 - JTAG Active is disabled in the LP domain. 1 - JTAG Active is enabled in the LP domain.
0 CAAM_EN	CAAM Security Violation Enable This bit enables CAAM Security Violation Input. When set, a CAAM Security Violation causes an LP security violation, which clears LP sensitive data. 0 - CAAM Security Violation is disabled in the LP domain. 1 - CAAM Security Violation is enabled in the LP domain.

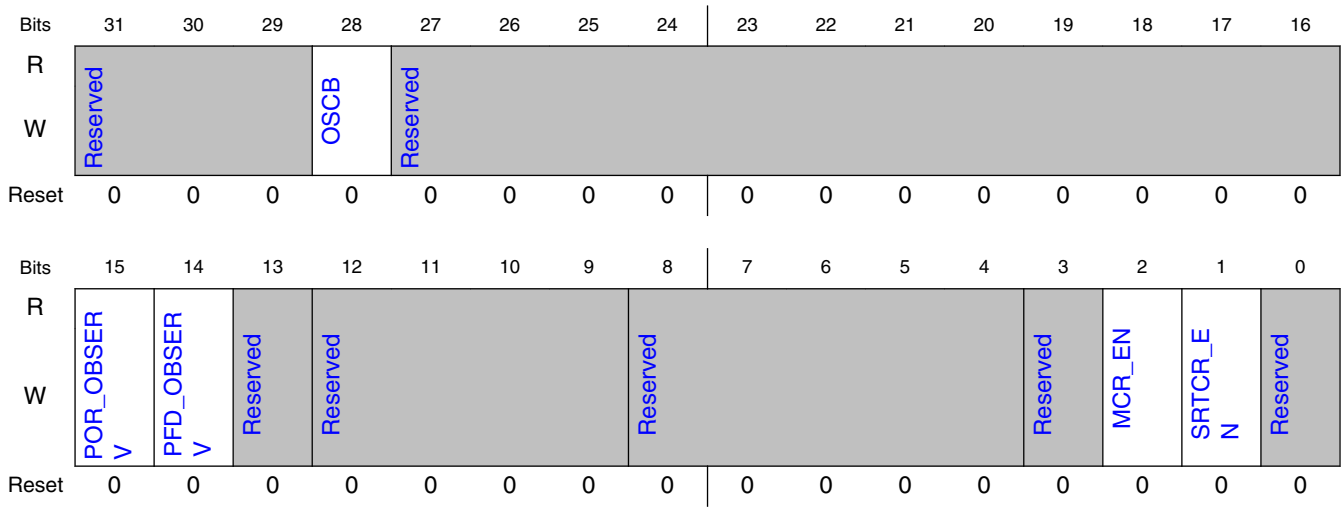
11.7.19 SNVS_LP Security Events Configuration Register (LPSECR)

The SNVS_LP Security Events Configuration Register is used to configure security event detection in the LP section. This register cannot be programmed when LPSECR is locked for write. This is a privileged write register.

11.7.19.1 Offset

Register	Offset
LPSECR	48h

11.7.19.2 Diagram



11.7.19.3 Fields

Field	Description
31-29 —	Reserved
28 OSCB	Oscillator Bypass When OSCB=1 the osc_bypass signal is asserted. That signal asks SoC logic external to SNVS to bypass the SoC's normal SRTC clock source and drive the SRTC clock from an alternate source. 0 - Normal SRTC clock oscillator not bypassed. 1 - Normal SRTC clock oscillator bypassed. Alternate clock can drive the SRTC clock source.
27-16 —	Reserved
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. 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 —	Reserved
12-9 —	Reserved

Table continues on the next page...

Field	Description
8-4 —	Reserved
3 —	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 SRTCR_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 —	Reserved

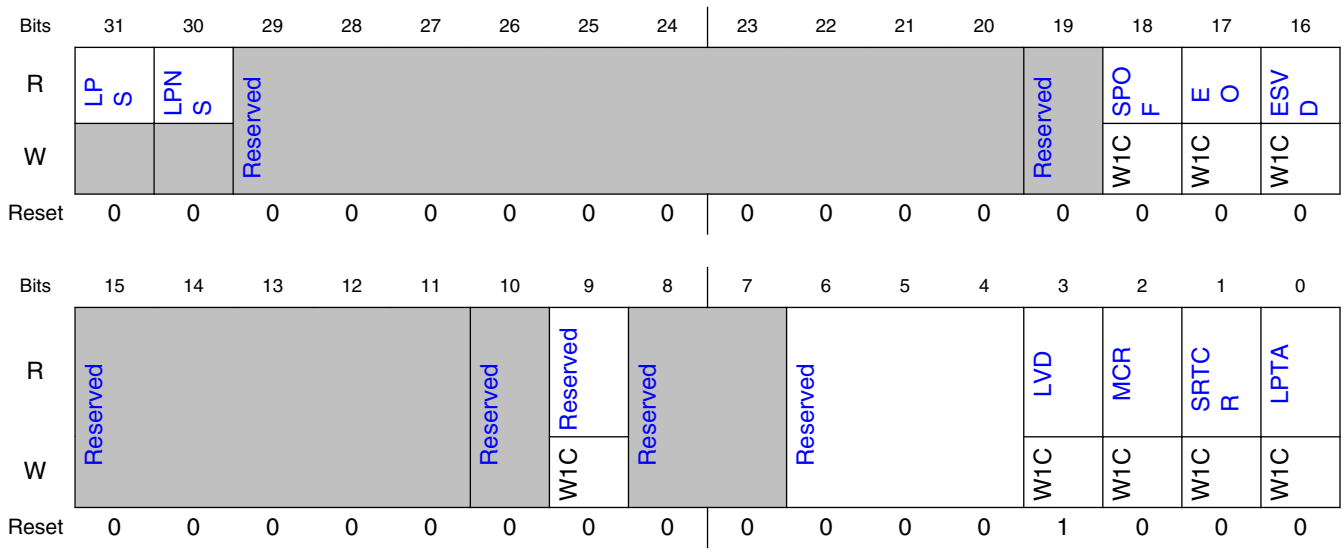
11.7.20 SNVS_LP Status Register (LPSR)

The SNVS_LP Status Register reflects the internal state and behavior of the SNVS_LP. This is a privileged write register.

11.7.20.1 Offset

Register	Offset
LPSR	4Ch

11.7.20.2 Diagram



11.7.20.3 Fields

Field	Description
31 LPS	LP Section is Secured 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. 0 - LP section was not programmed in secure or trusted state. 1 - LP section was programmed in secure or trusted state.
30 LPNS	LP Section is Non-Secured 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. 0 - LP section was not programmed in the non-secure state. 1 - LP section was programmed in the non-secure state.
29-20 —	Reserved
19 —	Reserved
18 SPOF	Set Power Off The SPO bit is set when the power button is pressed longer than the configured debounce time. Writing to the SPO bit will clear the set_pwr_off_irq interrupt.

Table continues on the next page...

Field	Description
	0 - Set Power Off was not detected. 1 - Set Power 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 —	Reserved
10 —	Reserved
9 —	Reserved
8-7 —	Reserved
6-4 —	Reserved
3 LVD	Digital Low Voltage Event Detected 0 - No low voltage event detected. 1 - Low voltage event is detected.
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.

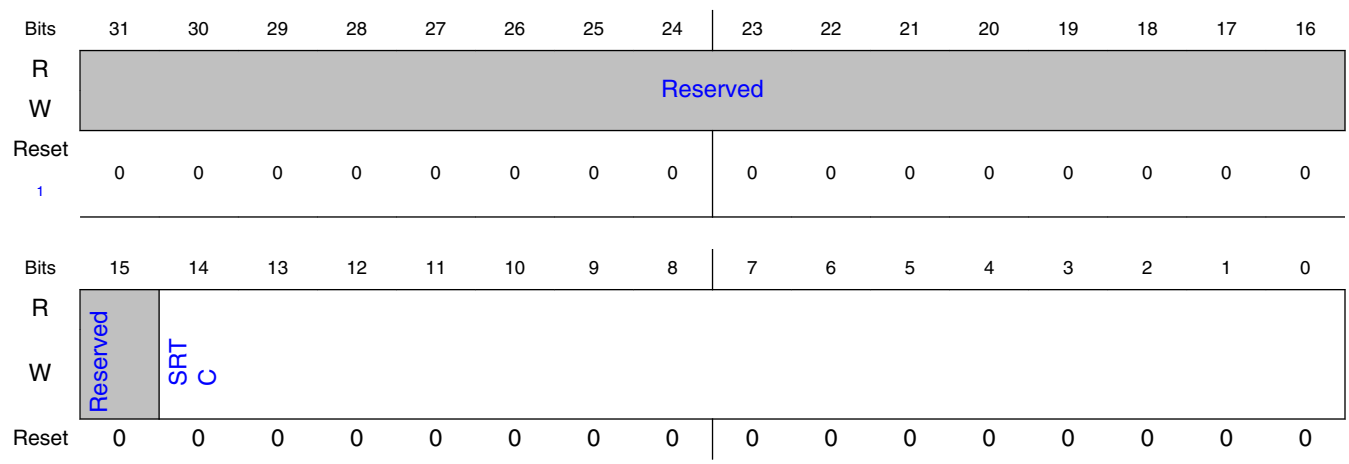
11.7.21 SNVS_LP Secure Real Time Counter MSB Register (LPSRTCMR)

The SNVS_LP Secure Real Time Counter MSB register contains the 15 most-significant bits of the LP Secure Real Time Counter. This is a privileged write register.

11.7.21.1 Offset

Register	Offset
LPSRTCMR	50h

11.7.21.2 Diagram



1. This register is reset at LP POR, but it is not reset by an LP software reset (i.e. writing a 1 to LP_SWR in HPCOMR).

11.7.21.3 Fields

Field	Description
31-15 —	Reserved
14-0 SRTC	LP Secure Real Time Counter The most-significant 15 bits of the SRTC. 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.

11.7.22 SNVS_LP Secure Real Time Counter LSB Register (LPSRTCLR)

The SNVS_LP Secure Real Time Counter LSB register contains the 32 least-significant bits of the secure real time counter. This is a privileged write register.

11.7.22.1 Offset

Register	Offset
LPSRTCLR	54h

11.7.22.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SRTC															
Reset [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SRTC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This register is reset at LP POR, but it is not reset by an LP software reset (i.e. writing a 1 to LP_SWR in HPCOMR).

11.7.22.3 Fields

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.

11.7.23 SNVS_LP Time Alarm Register (LPTAR)

The SNVS_LP Time Alarm register contains the 32-bit LP Time Alarm value. This is a privileged write register.

11.7.23.1 Offset

Register	Offset
LPTAR	58h

11.7.23.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LPTA															
Reset [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LPTA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This register is reset at LP POR, but it is not reset by an LP software reset (i.e. writing a 1 to LP_SWR in HPCOMR).

11.7.23.3 Fields

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).

11.7.24 SNVS_LP Secure Monotonic Counter MSB Register (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. This is a non-privileged read-only register.

11.7.24.1 Offset

Register	Offset
LPSMCMR	5Ch

11.7.24.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MC_ERA_BITS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MON_COUNTER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.24.3 Fields

Field	Description
31-16 MC_ERA_BITS	Monotonic Counter Era Bits These bits are inputs to the module and typically connect to fuses. When the Monotonic Counter is in use (i.e. enabled and valid and powered by an uninterrupted power source), and the boot software detects that the Monotonic Counter most-significant 16 Bits and Monotonic Counter LSB Register have been reset (MC_ENV=0), the boot software can take action to ensure that the value in the monotonic counter remains monotonic (i.e. never decreasing). The action is to blow an additional MC_ERA_BITS fuse. Since the MC_ERA_BITS field forms the most-significant field of the monotonic counter, blowing an additional fuse guarantees that the new monotonic counter value is higher than any previous value. Since the Monotonic Counter is reset on an LP Software Reset, an excessive number of MC_ERA_BITS fusez may be consumed if LP Software Reset is used repeatedly.
15-0 MON_COUNTER	Monotonic Counter most-significant 16 Bits **Note that writing to this register does *not* change the value of this field to the value that was written.**

Field	Description
	The 48-bit monotonic counter value (consisting of LPSMCMR[MON_COUNTER] prepended to LPSMCLR[MON_COUNTER]) is incremented by one when: • A write transaction to the LPSMCMR or LPSMCLR register is detected. • The MC_ENV bit is set. • MC_SL and MC_HL bits are not set. This value can be reset only by LP software reset or LP POR.

11.7.25 SNVS_LP Secure Monotonic Counter LSB Register (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. This is a non-privileged read-only register.

11.7.25.1 Offset

Register	Offset
LPSMCLR	60h

11.7.25.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MON_COUNTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MON_COUNTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.25.3 Fields

Field	Description
31-0 MON_COUNTER	Monotonic Counter bits **Note that writing to this register does *not* change the value of this field to the value that was written.** **The 48-bit monotonic counter value (consisting of LPSMCMR[MON_COUNTER] prepended to LPSMCLR[MON_COUNTER]) is incremented by one when:** • A write transaction to the LPSMCMR or LPSMCLR register is detected. • The MC_ENV bit is set. • MC_SL and MC_HL bits are not set. This value can be reset only by LP software reset or LP POR.

11.7.26 SNVS_LP Digital Low-Voltage Detector Register (LPLVDR)

The SNVS_LP Digital Low-Voltage Detector Register is a 32-bit read/write register that is used for storing the low-voltage detector value, as described in Digital Low-Voltage Detector (LVD). This is a privileged write register.

11.7.26.1 Offset

Register	Offset
LPLVDR	64h

11.7.26.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LVD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LVD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.26.3 Fields

Field	Description
31-0 LVD	Low-Voltage Detector Value

11.7.27 SNVS_LP General Purpose Register 0 (legacy alias) (LPGPR0_legacy_alias)

See register SNVS_LP General Purpose Registers 0 .. 3 (LPGPR0 - LPGPR3).

11.7.27.1 Offset

Register	Offset
LPGPR0_legacy_alias	68h

11.7.27.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	GPR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	GPR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.27.3 Fields

Field	Description
31-0 GPR	General Purpose Register When GPR_SL or GPR_HL bit is set, the register cannot be programmed.

11.7.28 SNVS_LP Zeroizable Master Key Register (LPZMKR0 - LPZMKR7)

The SNVS_LP Zeroizable Master Key Registers contain the 256-bit zeroizable master key value. This is a privileged write register. 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.

11.7.28.1 Offset

For a = 0 to 7:

Register	Offset
LPZMKRa	6Ch + (a × 4h)

11.7.28.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ZMK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ZMK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.28.3 Fields

Field	Description
31-0 ZMK	Zeroizable Master Key Each of these registers contains 32 bits of the 256-bit ZMK value. The least-significant byte is at offset 6Ch.

11.7.29 SNVS_LP General Purpose Registers 0 .. 3 (LPGPR0 - LPGPR3)

The SNVS_LP General Purpose Register is a 128-bit read/write register located in SNVS_LP, which can be used by any application for retaining data during an SoC power-down mode. This is a privileged read/write register. The full GPR register is accessed as 4 32-bit registers located in successive word addresses starting at offset 90h. For backward compatibility with earlier versions of SNVS, LPGPR0 is also aliased at its original offset of 68h. New software should access the GPR register at the preferred offset of 90h.

11.7.29.1 Offset

Register	Offset
LPGPR0	90h

Table continues on the next page...

Register	Offset
LPGPR1	94h
LPGPR2	98h
LPGPR3	9Ch

11.7.29.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	GPR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	GPR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.7.29.3 Fields

Field	Description
31-0 GPR	General Purpose Register When GPR_SL or GPR_HL bit is set, the register cannot be programmed.

11.7.30 SNVS_HP Version ID Register 1 (HPVIDR1)

The SNVS_HP Version ID Register 1 is a non-privileged 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.

11.7.30.1 Offset

Register	Offset
HPVIDR1	BF8h

11.7.30.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IP_ID															
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAJOR_REV								MINOR_REV							
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

11.7.30.3 Fields

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

11.7.31 SNVS_HP Version ID Register 2 (HPVIDR2)

The SNVS_HP Version ID Register 2 is a non-privileged read-only register that indicates the current version of the SNVS. Version ID register 2 consists of the following fields: integration options, ECO revision, and configuration options.

11.7.31.1 Offset

Register	Offset
HPVIDR2	BFCh

11.7.31.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IP_ERA								INTG_OPT							
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ECO_REV								CONFIG_OPT							
W																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

11.7.31.3 Fields

Field	Description
31-24 IP_ERA	IP Era 00h - Era 1 or 2 03h - Era 3 04h - Era 4 05h - Era 5 06h - Era 6
23-16 INTG_OPT	SNVS Integration Options
15-8 ECO_REV	SNVS ECO Revision
7-0 CONFIG_OPT	SNVS Configuration Options

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 Holdings.
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.
CFB*n*	Cipher FeedBack An encryption mode of operation. CFB128 is one of the official modes of operation specified for AES.CFB8 is the official mode of operation specified for DES.
CHA	Cryptographic Hardware Accelerator One of the hardware accelerators used in CAAM
CRJD	Control Replacement Job Descriptor
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

Table continues on the next page...

Table A-1. Acronyms and abbreviated terms (continued)

Term	Meaning
DEK	Data Encryption Key.
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.
DRBG	Deterministic Random Bit Generator A deterministic algorithm that generates a sequence of numbers whose values are statistically random. Sometimes called "PRNG" (pseudorandom number generator).
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 consisting of the byte 36h repeated 64 times for MD5, SHA-1, SHA-224 or SHA-256, and reported 128 times for SHA-384 or SHA-512
ICV	Integrity Check Value A checksum or message digest that allows detection of errors or changes in data.
IJD	Inline Job Descriptor
IV	Initialization Vector A value used to initialize some encryption modes of operation
JD	Job Descriptor
JDKEK	Job Descriptor Key Encryption Key
JQC	Job Queue Controller The hardware that schedules jobs received from the Job Rings and RTIC
JR	Job Ring
KMOD	Key Modifier (field in SMAP register)
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.
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 consisting of the byte 5Ch repeated 64 times for MD5, SHA-1, SHA-224 or SHA-256, and reported 128 times for SHA-384 or SHA-512
OTPMK	One-time-programmable Master Key

Table continues on the next page...

Table A-1. Acronyms and abbreviated terms (continued)

Term	Meaning
PKHA	Public Key Hardware Accelerator (ECC, RSA, DH, DSA)
POR	Power On Reset.
PRNG	Pseudo Random Number Generator A deterministic algorithm that generates a sequence of numbers whose values are statistically random. See DRBG.
PSP	Public Security Parameter Security-related public information whose modification can compromise the security of a cryptographic module.
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 DRBG.
RJD	Replacement Job Descriptor
RTIC	Run-Time Integrity Checker A logic block within CAAM that generates a security event if the integrity of selected memory areas have been compromised.
SD	Shared Descriptors
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)
SMAP	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.
TD	Trusted Descriptor
TDSK	Trusted Descriptor Signing Key
TDKEK	Trusted Descriptor Key Encryption Key
TRK	Trusted Root Key
ZMK	Zeroizable Master 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 register interface are designed to control accesses from multiple bus masters to CAAM register pages.The access control features built into CAAM's Secure Memory are designed to control accesses from multiple bus masters to Security Memory partitions. (See *Partition*.)
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 *Allocate Page* Command by means of the Secure Memory Command Register. (The phrase 'allocated to' is used interchangeably with 'claimed by'. Also see *Own*.
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 *Red Blob*.)
Black key	A key that has been encrypted using either the JDKEK or the TDKEK. (See *Red Key*.)
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 *Master Key*.)
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 *Allocate Page* Command by means of Secure Memory Command Register. (The phrase "claimed by" is used interchangeably with "allocated to". See also *Own*.
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 *CSP*, 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, a private key used for signing data, a public key used for verifying a signature, a private or public key used in a key establishment operation, 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

Table continues on the next page...

Table A-2. Glossary of terms (continued)

Term	Description
	unallocated pages. A processor de-allocates a page associated with a partition that the processor owns by issuing a *De-allocate Page* 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.
Derived HMAC key	A performance improving structure consisting of the HMAC key, XORed with IPAD, and the same HMAC key XORed with OPAD, both then processed with the underlying HASH function.
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 *Import*)
Fail mode	CAAM clears its CSP registers (e.g. key registers) upon entrance to *Fail Mode*. CAAM enters Fail Mode when the SNVS's security state machine enters its Fail State. This could be due to the detection of scan or JTAG testing, or due to the failure of a security module.
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 *SM Blob Key Encryption Key, Master Key*.)
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 *Export*)
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 *Trusted Descriptor Key Encryption Key*.
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 *JDKEK, TDKEK*)
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 DID.
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.

Table continues on the next page...

Table A-2. Glossary of terms (continued)

Term	Description
Master key	The master key is a 256-bit secret value that CAAM receives from the SNVS. (See *Non-volatile Test Key*, *OTP Master Key*, *Zeroizable Master Key*, *Secure Memory Blob Key Encryption Key*, and General Memory *Blob Key Encryption Key*.)
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 (all 0s) 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 *Claim*)
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 *Partition*)
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 *Ownership*)
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.
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 *Black Blob*)
Red key	A key that is not encrypted. (See *Black Key*)
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 *Claim, Own, Partition, Page*)
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.

Table continues on the next page...

Table A-2. Glossary of terms (continued)

Term	Description
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 *General Memory Blob Key Encryption Key, Master Key*)
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 an uninterrupted power source 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)]
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 faciliate 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.
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 *Trusted Descriptor Signing Key*)
Trusted descriptor signing key	The *Trusted Descriptor Signing Key* (TDSK) is a key used to sign and verify the signature over trusted descriptors. A new TDSK value is generated by the CAAM 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 JRaDID 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 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 *Job Descriptor Key Encryption Key*)
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 Non-Secure 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.

Table continues on the next page...

Table A-2. Glossary of terms (continued)

Term	Description
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 '*unallocated*' is used interchangeably with the term '*unclaimed*'.
Unclaimed	The term 'unclaimed' is used interchangeably with the term 'unallocated'.
Word	A word of memory or a one-word register contains 32 bits.
Zeroizable master key	The Zeroizable Master Key (ZMK) 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 ZMK 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 security violation is detected.(See *Master Key*.)
Zeroize	A set of data storage locations is *zeroized* by overwriting the storage locations with a value (not necessarily 0) that is independent of the previous content of the storage locations.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 that may be provided in NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

Document Number IMX8MNSRM
Revision 0, 01/2020